SEVENTH FRAMEWORK PROGRAMME
Research Infrastructures

INFRA-2010-2.3.1 – First Implementation Phase of the European High Performance Computing (HPC) service PRACE

PRACE-1IP

PRACE First Implementation Project

Grant Agreement Number: RI-261557

D7.4.2
Benchmarking and Performance Modelling on Tier-0 Systems

Final

Version: 1.0
Authors: Mark Bull (EPCC), Stefanie Janetzko (FZJ), Jose Carlos Sancho (BSC), Jeroen Engelberts (SARA)
Date: 26.03.2012

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: RI-261557**	
	Project Title: PRACE First Implementation Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: D7.4.2	
	Deliverable Nature: DOC_TYPE: Report	
	Deliverable Level: PU / PP / RE / CO *	**Contractual Date of Delivery:** 31 / March / 2012
		Actual Date of Delivery: 31 / March / 2012
	EC Project Officer: Thomas Reibe	

* - The dissemination level are indicated as follows: **PU** – Public, **PP** – Restricted to other participants (including the Commission Services), **RE** – Restricted to a group specified by the consortium (including the Commission Services). **CO** – Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	**Title: Benchmarking and Performance Modelling on Tier-0 Systems**	
	ID: <**D7.4.2**>	
	Version: <1.0>	**Status:** Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2007	
	File(s): D7.4.2.docx	
Authorship	**Written by:**	Stefanie Janetzko, FZJ Mark Bull, EPCC Jose Carlos Sancho, BSC Jeroen Engelberts, SARA
	Contributors:	Carlo Cavazzoni, CINECA Marios Chatziangelou, GRNET Jacques David, CEA Raul de la Cruz, BSC Dimitris Dellis, GRNET John Donners, SARA Jussi Enkovaara, CSC Xu Guo, EPCC Soon-Heum Ko, LiU Jesus Labarta, BSC Fernando Nogueira, UC-LCA Micael Oliveira, UC-LCA Andrew Porter, STFC Xavier Saez, BSC Carlos Simoes, UC-LCA Andrew Sunderland, STFC
	Reviewed by:	Dietmar Erwin, FZJ Gabriele Carteni, BSC
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	13/02/2012	Draft	Structure, results of benchmark runs, general benchmark sections
0.2	29/02/2012	Draft	Added further benchmark information and analysis from BCOs
0.3	02/03/2012	Draft	Added performance modelling section, ES and Introduction
0.4	12/03/2012	Draft	Added synthetic benchmark section, finalization and proof reading
0.5	23/03/2012	Draft	Updated version after project internal review
1.0	26/03/2012	Final version	

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, applications requirements, benchmarking, performance modelling.

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] *Final Benchmark Suite*, PRACE Preparatory Phase Deliverable 6.3.2.

[2] *Report on the Application Benchmarking Results of Prototype Systems,* PRACE Preparatory Phase Deliverable 5.4.

[3] Frings, W.; Schnurpfeil, A.; Meier, S.; Janetzko, F.; Arnold, L. (2010). *A Flexible, Application- and Platform-Independent Environment for Benchmarking* in Parallel Computing: From Multicores and GPU's to Petascale, ed.: B. Chapman, F. Desprez, G.R. Joubert, A. Lichnewsky, F. Peters and T. Priol, Amsterdam, IOS Press, 2010. Advances in Parallel Computing Volume 19. - 978-1-60750-529-7. - S. 423 – 430

[4] Houzeaux, G., de la Cruz, R., Vazquez, M., *Parallel Uniform Mesh Subdivision in Alya* PRACE Whitepaper at www.prace-ri.eu

[5] *Technical Assessment Report of Prototype Systems*, PRACE Preparatory Phase Deliverable 5.2.

[6] *Assessment report on communication and I/O infrastructure of prototype systems,* PRACE Preparatory Phase Deliverable 5.3.

[7] Pillret, V., Labarta, J., Cortes, T., Girona, S.,: *Paraver: A tool to visualize and analyze parallel code,* in WoTUG-18 (1995), pp. 17–31.

[8] Kojak. http://www.fz-jeulick.de/zam/kojak/, 2006.
[9] Wolf, F., Wylie, B., Abraham, E., Becker, D., Frings, W., Fűrlinger, K., Geimer, M., Hermanns, M., Mohr, B., Moore, S., Pfeifer, M., and Szebenyi, Z. *Usage of the SCALASCA toolset for scalable performance analysis of large-scale parallel applications*, in Proc. of the 2nd HLRS Parallel Tools Workshop (July 2008), Springer, pp. 157–167.
[10] Brunst, H., Kranzlműller, D., and Nagel, W. E., *Tools for scalable parallel program analysis - Vampir NG and DeWiz,* in Distributed and Parallel Systems, Cluster and Grid Computing 777 (2004).
[11] Shende, S., and Malony, A. D. *The TAU parallel performance system,* in The International Journal of High Performance Computing Applications 20, 2 (Summer 2006), 287–331.
[12] Taylor, V., Wu, X., and Stevens, R., *PROPHESY: A Web-based Performance Analysis and Modeling System for Parallel and Distributed Applications,* Performance TOOLS 2003 (Tool demonstrations), Urbana, Illinois, September 2-5, 2003.
[13] S. Girona, T. Cortes and J. Labarta , *Analyzing scheduling policies using DIMEMAS,* Environments and Tools for Parallel Scientific Computing III, Faverges de la Tour, France, August 1996.
[14] EXTRAE, available at
http://www.bsc.es/computer-sciences/performance-tools/downloads

List of Acronyms and Abbreviations

API	Application Programming Interface
BAdW	Bayerischen Akademie der Wissenschaften (Germany)
BCO	Benchmark Code Owner
BLAS	Basic Linear Algebra Subprograms
BSC	Barcelona Supercomputing Center (Spain)
CAF	Co-Array Fortran
CEA	Commissariat à l'Energie Atomique (represented in PRACE by GENCI, France)
CINECA	Consorzio Interuniversitario, the largest Italian computing centre (Italy)
CINES	Centre Informatique National de l'Enseignement Supérieur (represented in PRACE by GENCI, France)
CPU	Central Processing Unit
CSC	Finnish IT Centre for Science (Finland)
CSCS	The Swiss National Supercomputing Centre (represented in PRACE by ETHZ, Switzerland)
CUDA	Compute Unified Device Architecture (NVIDIA)
DEISA	Distributed European Infrastructure for Supercomputing Applications. EU project by leading national HPC centres.
DGEMM	Double precision General Matrix Multiply
DMA	Direct Memory Access
DNA	DeoxyriboNucleic Acid
DP	Double Precision, usually 64-bit floating point numbers
DRAM	Dynamic Random Access memory
EC	European Community
EESI	European Exascale Software Initiative
EoI	Expression of Interest
EP	Efficient Performance, e.g., Nehalem-EP (Intel)

EPCC	Edinburg Parallel Computing Centre (represented in PRACE by EPSRC, United Kingdom)
EPSRC	The Engineering and Physical Sciences Research Council (United Kingdom)
ETHZ	Eidgenössische Technische Hochschule Zuerich, ETH Zurich (Switzerland)
ESFRI	European Strategy Forum on Research Infrastructures; created roadmap for pan-European Research Infrastructure.
EX	Expandable, e.g., Nehalem-EX (Intel)
FFT	Fast Fourier Transform
FP	Floating-Point
FPGA	Field Programmable Gate Array
FPU	Floating-Point Unit
FZJ	Forschungszentrum Jülich (Germany)
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GCS	Gauss Centre for Supercomputing (Germany)
GÉANT	Collaboration between National Research and Education Networks to build a multi-gigabit pan-European network, managed by DANTE. GÉANT2 is the follow-up as of 2004.
GENCI	Grand Equipement National de Calcul Intensif (France)
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GigE	Gigabit Ethernet, also GbE
GLSL	OpenGL Shading Language
GNU	GNU's not Unix, a free OS
GPGPU	General Purpose GPU
GPU	Graphic Processing Unit
GS	Gram-Schmidt
HDD	Hard Disk Drive
HE	High Efficiency
HMM	Hidden Markov Model
HMPP	Hybrid Multi-core Parallel Programming (CAPS enterprise)
HP	Hewlett-Packard
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing
HPCC	HPC Challenge benchmark, http://icl.cs.utk.edu/hpcc/
HPL	High Performance LINPACK
HT	HyperTransport channel (AMD)
HWA	HardWare accelerator
IB	InfiniBand
IBA	IB Architecture
IBM	Formerly known as International Business Machines
ICE	(SGI)
IDRIS	Institut du Développement et des Ressources en Informatique Scientifique (represented in PRACE by GENCI, France)
IEEE	Institute of Electrical and Electronic Engineers
IESP	International Exascale Project
IL	Intermediate Language

IMB	Intel MPI Benchmark
I/O	Input/Output
IOR	Interleaved Or Random
JSC	Jülich Supercomputing Centre (FZJ, Germany)
JuBE	Jülich Benchmarking Environment
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
KTH	Kungliga Tekniska Högskolan (represented in PRACE by SNIC, Sweden)
LBE	Lattice Boltzmann Equation
LINPACK	Software library for Linear Algebra
LLNL	Laurence Livermore National Laboratory, Livermore, California (USA)
LQCD	Lattice QCD
LRZ	Leibniz Supercomputing Centre (Garching, Germany)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MD	Molecular Dynamic
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MGS	Modified Gram-Schmidt
MHz	Mega (= 10^6) Hertz, frequency =10^6 periods or clock cycles per second
MIPS	Originally Microprocessor without Interlocked Pipeline Stages; a RISC processor architecture developed by MIPS Technology
MKL	Math Kernel Library (Intel)
ML	Maximum Likelihood
Mop/s	Mega (= 10^6) operations per second (usually integer or logic operations)
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
MPP	Massively Parallel Processing (or Processor)
MPT	Message Passing Toolkit
mxm	DP matrix-by-matrix multiplication mod2am of the EuroBen kernels
NAS	Network-Attached Storage
NCF	Netherlands Computing Facilities (Netherlands)
NDA	Non-Disclosure Agreement. Typically signed between vendors and customers working together on products prior to their general availability or announcement.
NoC	Network-on-a-Chip
NFS	Network File System
OpenCL	Open Computing Language
OpenGL	Open Graphic Library
Open MP	Open Multi-Processing
OS	Operating System
PABS	PRACE Application Benchmark Suite
PGAS	Partitioned Global Address Space
PGI	Portland Group, Inc.
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
PRACE-PP	PRACE Preparatory Project
PSNC	Poznan Supercomputing and Networking Centre (Poland)
QCD	Quantum Chromodynamics
QCDOC	Quantum Chromodynamics On a Chip
QPACE	QCD Parallel Computing on the Cell

QR	QR method or algorithm: a procedure in linear algebra to compute the eigenvalues and eigenvectors of a matrix
SARA	Stichting Academisch Rekencentrum Amsterdam (Netherlands)
SDK	Software Development Kit
SGEMM	Single precision General Matrix Multiply, subroutine in the BLAS
SGI	Silicon Graphics, Inc.
SIMD	Single Instruction Multiple Data
SMP	Symmetric MultiProcessing
SNIC	Swedish National Infrastructure for Computing (Sweden)
SP	Single Precision, usually 32-bit floating point numbers
STF	Self-consistent field
STFC	Science and Technology Facilities Council (represented in PRACE by EPSRC, United Kingdom)
SVN	Apache Subversion
TB	Tera (= 240 ~ 1012) Bytes (= 8 bits), also TByte
TGCC	Très Grand Centre de calcul du CEA, CURIE's installation site
TFlop/s	Tera (= 1012) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
UFM	Unified Fabric Manager (Voltaire)
UNICORE	Uniform Interface to Computing Resources. Grid software for seamless access to distributed resources.
UPC	Unified Parallel C

Executive Summary

This document is a report on the activities of Task 7.4 “Applications requirements for Tier-0 systems” in the areas of benchmarking and performance modelling. We describe the work undertaken in updating the PRACE Application Benchmark Suite (PABS) developed in the PRACE Preparatory Phase project, and the results of porting and running these benchmarks on the two available Tier-0 systems (JUGENE and CURIE). We also report on the results of running the collection of synthetic benchmarks gathered by the Preparatory Phase project on CURIE. Finally, we illustrate the potential benefits of performance modelling in capturing application requirements by developing performance models for two of the PABS application codes, and assessing the likely impact of architecture changes on their performance.

1 Introduction

1.1 Background and Purpose

Task 7.4 has the mission to assess the requirements of applications for Tier-0 systems. One of the principal ways of achieving this is via benchmarking, i.e. by collecting, maintaining and running a set for application and synthetic benchmarks. The PRACE Preparatory Phase project undertook a significant amount of work to identify, port, run, and analyse the results of a large collection of such benchmarks. This deliverable reports on the continuation of this work, including updating and maintaining the benchmark suites, and gathering and analysing further performance data on Tier-0 systems using them. However, we recognise that benchmarking has its limitations: it cannot say very much about how applications will perform on future systems whose design may be known, or anticipated, but to which access is not yet possible. A related technique which addresses this problem is performance modelling. To illustrate how this works, we selected two of the PABS application codes and construct detailed performance models for them. This not only gives additional insight into their behaviour, but allows us to make predictions of how their performance might respond to future changes in architectural parameters such as CPU speed, network latency and network bandwidth.

1.2 Objectives

The objectives of this deliverable are as follows:

- to provide a status report on the PRACE Application Benchmark Suite,
- to provide and analyse both application and synthetic benchmark results for the available Tier-0 systems, and
- to demonstrate the viability of performance modelling as a tools for capturing application requirements for future Tier-0 systems.

1.3 Dependencies

In PRACE-1IP, WP8 may draw on the information contained in this deliverable to assist in the assessment of suitable architectures for Tier-0 systems. In PRACE-2IP, Task 7.4 may

draw on the information to assist in the selection of applications for the Unified European Application Benchmark Suite.

1.4 Structure of the Document

The remainder of this document is structured as follows: Section 2 describes the work done to update, port, and maintain the PRACE Application Benchmark Suite (PABS), and Section 3 reports the results of running these benchmarks on the two available Tier-0 systems. Section 4 describes the synthetic benchmarks used in this task and reports the results of running these benchmarks on one of the Tier-0 systems. Section 5 contains an introduction to performance modelling and case studies of modelling two of the PABS application codes. Section 6 draws some conclusions and suggests future work. In Annex A (chapter 7) the raw data of the results of the PABS runs are shown and in Annex B (chapter 8) the list of PRACE Benchmark Code Owners can be found.

1.5 Audience

The intended audience for this deliverable includes:

- Users of Tier-0 systems who may wish to decide which Tier-0 system to run their applications on.
- Task 7.4 in PRACE-2IP, which will utilise the results to choose application codes for a Unified European Application Benchmark Suite.
- WP8 in PRACE-1IP, to assist in assessing architectures.
- Application developers who may which to consider performance modelling as an approach to characterising performance, and analysing requirements for future systems.

2 Application Benchmarking

One of the objectives of Task 7.4 is to analyse the application benchmarks produced by the PRACE Preparatory Phase on the Tier-0 systems[1]. Using these results it is possible to assist users in choosing the best system for their application. It was originally hoped that the benchmarks could be run on more Tier-0 systems, but the latest of these to come online (HERMIT at HLRS and SuperMUC at LRZ) were not available in time to obtain results for this deliverable. Nevertheless, the results obtained here will form a vital input to Task 7.4 of the Second Implementation Phase of PRACE, which is selecting a unified benchmark suite based on the existing PRACE and DEISA suites.

2.1 Overview

The work performed on the PRACE Application Benchmark Suite (PABS) is strongly related to the work performed in the PRACE Preparatory Project. In the Preparatory Project the benchmarks for the PRACE suite were selected and runs on the Prototypes were performed. In this deliverable, we follow closely the methodologies developed in the Preparatory Project[2].

The following work has been undertaken:

1. Update the codes in PABS to new versions if available (and stable), and create new test cases where possible to gain an up-to-date benchmark suite with potentially better scaling results.
2. Port the codes to the two Tier-0 systems which were available during the task lifetime, JUGENE and CURIE (also referred to as IBM BlueGene/P and Bull x86 Cluster in this document). The porting is not only necessary for the new Tier-0 machine CURIE, but also for JUGENE because of the updates performed on the codes and because not all applications ran on this machine in the Preparatory Phase.
3. Maintain PABS.
4. Perform and analyse the runs on the Tier-0 systems.

These steps are described in more detail in the following sections.

2.2 Updates and Porting

The following is a summary of the updating and porting work performed in PRACE-1IP WP7.4:

- **CODE_SATURNE**
  - Updated to new code version 2.0.1 and new datasets added
  - Ported to both systems
- **CP2K**
  - Updated to new code version 2.2.263 and new datasets added
  - Ported to both systems
- **CPMD**
  - Updated to new code version 3.15.1, new test case B
  - Ported to both systems, but runs only on CURIE because of memory constraints on JUGENE
- **EUTERPE**
  - No update necessary, version 2.54
  - Ported to both systems
- **GADGET**
  - No update necessary, version 2
  - Ported to both systems
- **GROMACS**
  - Updated to new version 4.5.4 and new datasets and system sizes added
  - Ported to both systems
- **NAMD**
  - Updated to new code version 2.8 and new dataset added
  - Ported to both systems
- **NEMO**
  - Updated to new code version 3.2.2 with MPI changes backported from 3.4, higher resolution setup and input data
  - Ported only to CURIE, because scaling is known to be low on the IBM BlueGene/P platform
- **NS3D**
  - No update was necessary, version 1.853
  - Ported to both systems
- **QCD**
  - No update was necessary, newest version (no version number)

  - Ported to both systems
- **QUANTUM_ESPRESSO**
  - Updated to new code version 4.3.1 and new larger dataset added
  - Ported to both systems
- **WRF**
  - No update was necessary, version 3.1.1
  - Ported to both systems
- **ALYA**
  - Updated to new code version (no version number) and new datasets added
  - Ported to both systems
- **AVBP**
  - Updated to new code version (no version number)
  - Ported to CURIE
- **BSIT**
  - Not done, because the code was originally designed for the Cell BE architecture, and porting expertise was not available. Furthermore, the test case is a weak-scaling task farm problem which does not give useful information on scalability.
- **ELMER**
  - Updated to new code version 6.1-Rev.4880 and new datasets
  - Ported only to CURIE, because problems on JUGENE with dynamic loading of libraries
- **GPAW**
  - Updated to new version 0.9.0-Rev.8581, new datasets and system sizes have been introduced
  - Ported to both systems
- **HELIUM**
  - No update was necessary (no version number)
  - Ported to both systems
- **OCTOPUS**
  - No update was necessary, version 3.2
  - Ported to both systems
- **PEPC**
  - Updated to new code version (version 2) and new larger test cases introduced
  - Ported to both systems, but scaling runs could only be performed on JUGENE, because of problems for larger runs with the current MPI version on CURIE.
- **SPECFEM3D**
  - Updated to new code version 5.1.1
  - Ported to CURIE
- **TRIPOLI_4**
  - Updated to new code version 4.7
  - Ported to CURIE

To summarise the porting status, 16 codes have been ported to both systems and five codes have been ported to CURIE only. One code from the code list of the Preparatory Project has been skipped.

For 14 codes, updates of the code version and input sets have been performed, which have made it possible to show scaling to higher core numbers (e.g. for GPAW and PEPC: see Chapter 3 for the results).

2.3 Maintaining PABS

The benchmark suite should be maintained for use by other work packages as well as by hosting sites. To make this possible all necessary data of a code is stored in the PRACE SVN server (a switch was performed in this task from the Preparatory Project Trac-system to a combination of SVN and wiki), this includes the source code and input sets as well as the configuration files for the benchmarking environment JuBE[3], as prepared already in the Preparatory Phase, which allows a smooth usage of PABS.

3 Application Benchmark Results

This chapter contains the results obtained on the Tier-0 systems available in the period of the project. These two systems are the IBM BlueGene/P (JUGENE) and the Bull x86 Cluster (CURIE). The JUGENE system consists of 73728 nodes, each of which contains four Power PC 450 850 MHz cores and 2GB of memory. The results reported here from CURIE are from the "fat node" partition, which contains 360 BULL S6010 nodes. Each node has four 8-core Nehalem EX 2.26 GHz processors, 128 GB of memory, and one link to a QDR Infiniband network. The methodology used is the same as in the Preparatory Phase Project: see Chapter 2 of [2] for details.

For each code we present the test cases and two different figures for each test case: first, a scaling plot, which is the execution time as a function of the number of CPUs (where “CPU” is used synonymously for “core”). Second, we show the performance (the reciprocal of execution time) per Peak-TFlop/s as function of the partition size in Peak-TFlop/s. In this figure the y-axis values are actually not meaningful but there are two reasons to show it: first, the behaviour of the curves is of interest: scaling is better if this curve does not decline too much, and ideal scaling is represented by a horizontal line. Secondly, the performance of codes can be compared between the two systems. Please note that it is not possible to compare different codes on one platform using this metric. To allow such a comparison we show two figures in the summary (Chapter 3.22), where the performance per Peak-TFlop/s is normalised to the 10 TFlop/s partition.

The corresponding tables of raw data are listed in the Annex A which can be found in Chapter 7. To each benchmark there was a Benchmark Code Owner (BCO) assigned. The BCO was responsible for the code including updates and porting as described in the previous chapter and the BCO also performed the runs and the analysis of the outcome as described in this chapter. The list of BCOs can be found in the Annex B in Chapter 8.

3.1 CODE_SATURNE

3.1.1 *Summary*

Code_Saturne is a general purpose CFD code, used for nuclear thermalhydraulics processes, coal and gas combustion, aeraulics, etc.

3.1.2 *Test Cases*

Test Case A consists of an isothermal Large Eddy Simulation in a T-junction, containing 10 million cells. Only the dynamics of the flow is investigated. The runs are carried out for 100 time steps, starting from an already developed flow.

Test Case B consists of a flow around the body of a submarine, consisting of 107 million cells.

3.1.3 *Results*

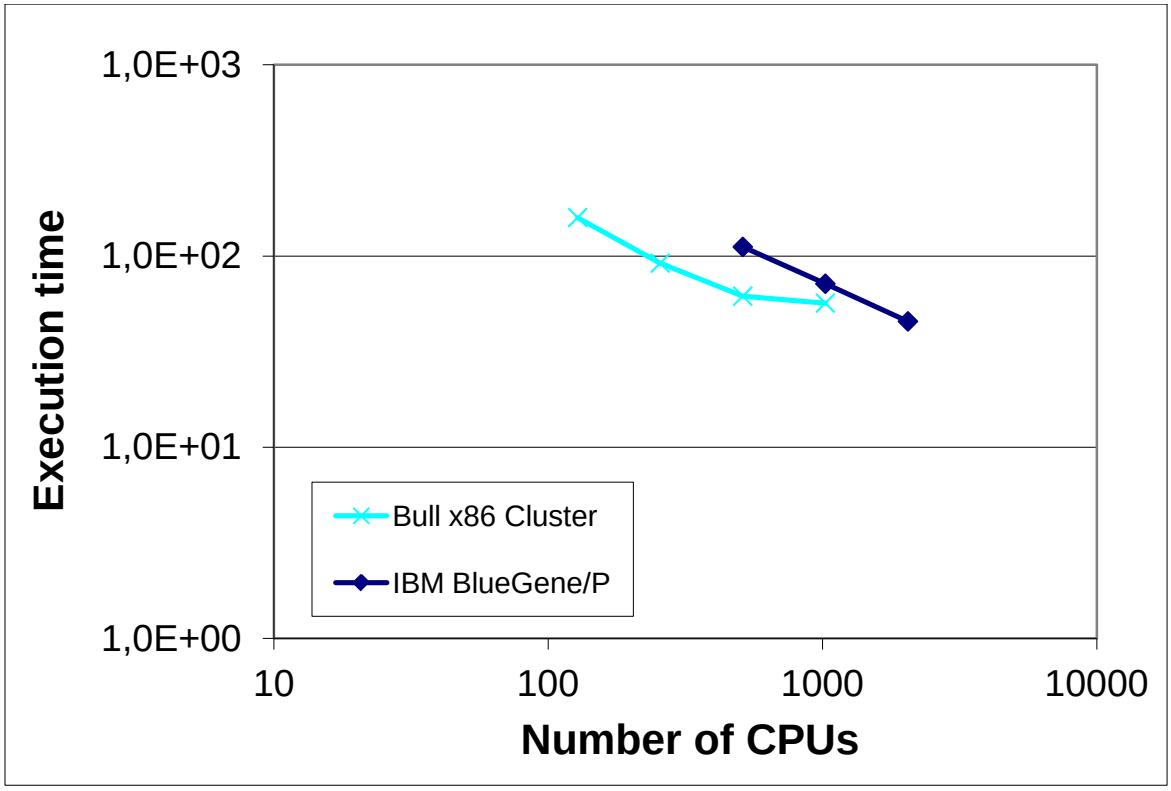

Figure 1: Execution time of Codes_Saturne, Test Case A

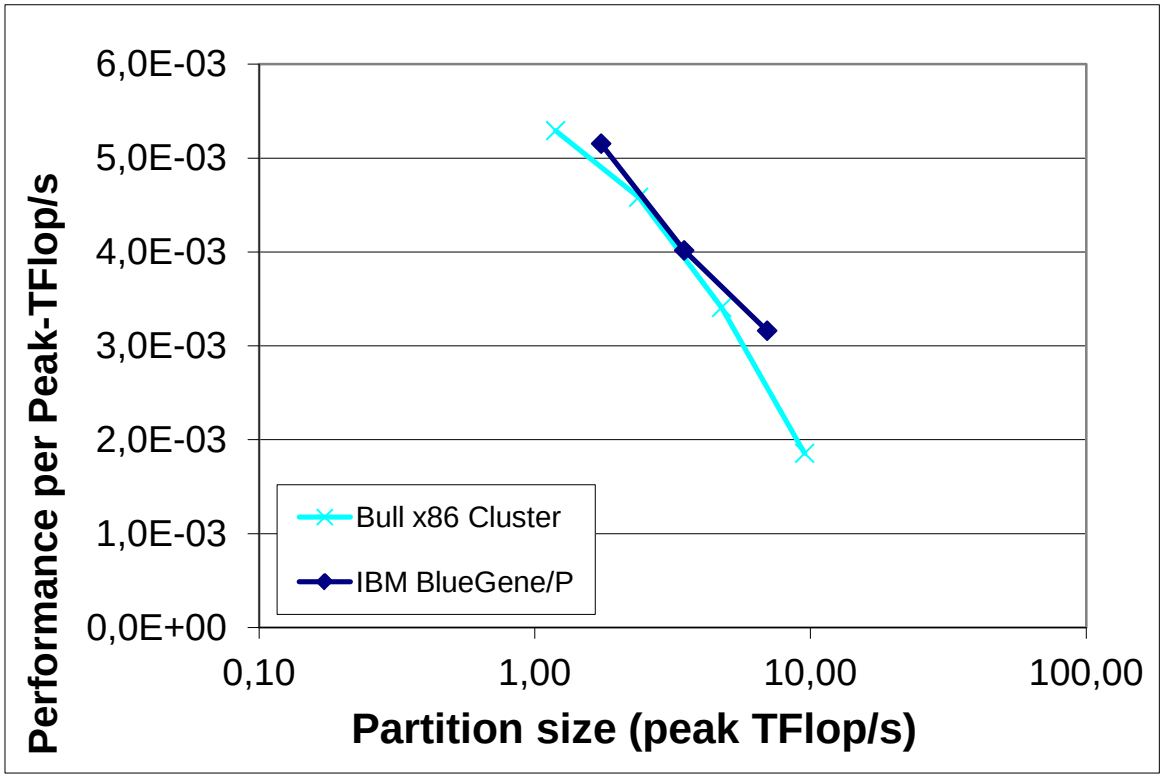

Figure 2: Performance per Peak-TFlop/s for Code_Saturne, Test Case A

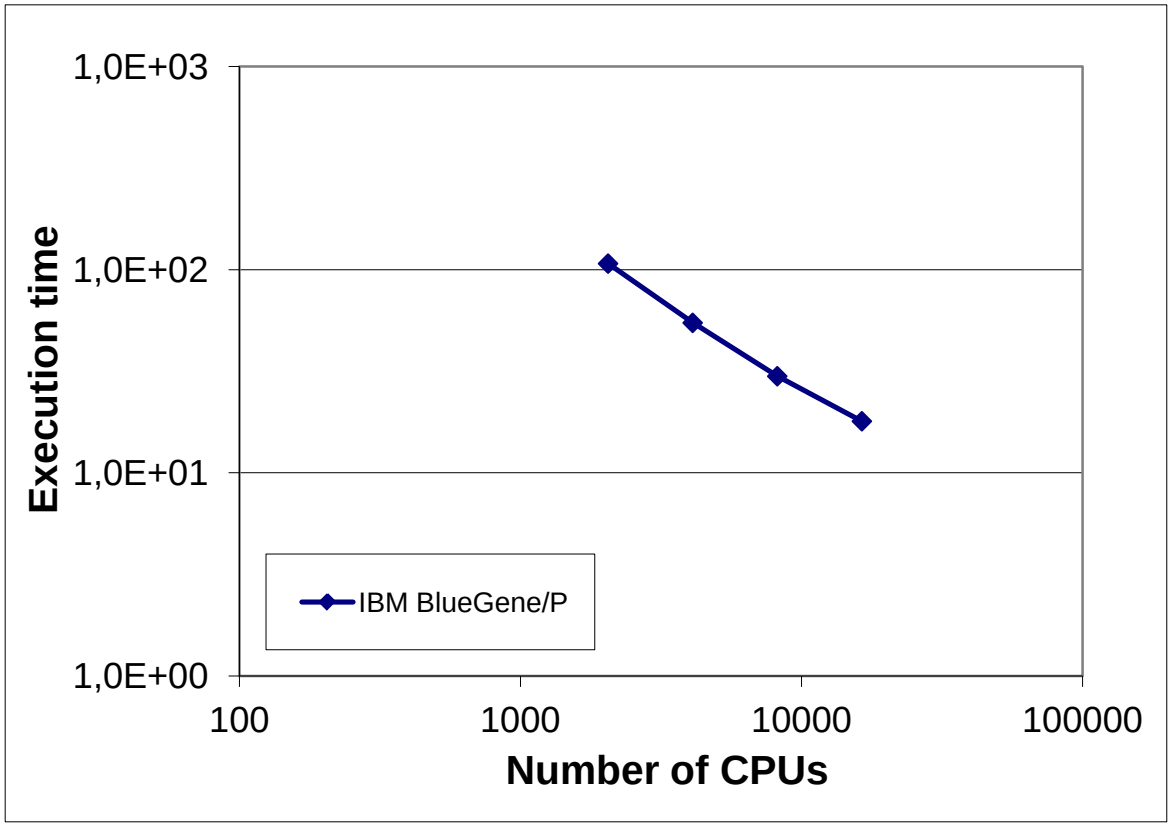

Figure 3: Execution time of Code_Saturne, Test Case B

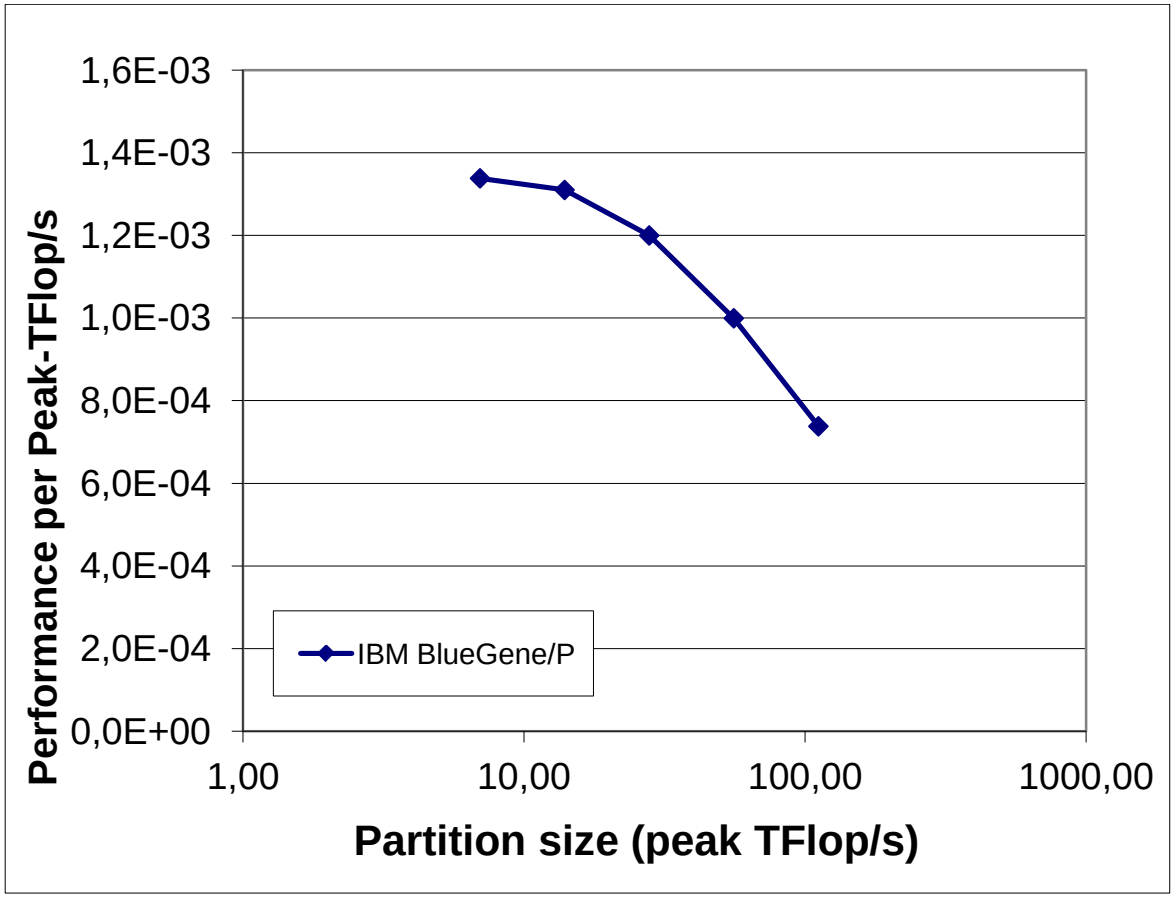

Figure 4: Performance per Peak-TFlop/s for Code_Saturne, Test Case B

3.1.4 *Analysis*

Initially optimized for IBM Blue Gene architectures, Code_Saturne generally performs well on a range of large-scale HPC architectures. Figure 4 demonstrates scaling out to 32768 cores of the IBM BlueGene/P when the dataset is suitably large (Case B - 107M cells). Performance benchmarks with the much smaller Case A (10M cells) scale very well to 512 cores of the Bull x86 Cluster and 4096 cores on the IBM BlueGene/P, but beyond this core count the communication costs of exchanging data across partitions begins to dominate due to the relatively small size of the dataset. Figure 2 shows that performance per peak TFlops/s is roughly equivalent on both the machines.

3.2 CP2K

3.2.1 *Summary*

CP2K is a community code to perform atomistic and molecular simulations of solid state, liquid, molecular and biological systems. It consists of several components for classical molecular dynamics and ab-initio density functional theory.

3.2.2 *Test Cases*

To minimize the benchmark time, in all cases the wavefunction was optimized with a number of separate jobs, and then used as restart in all subsequent runs.
Test Case A: 1 step Molecular Dynamics of 32 OMIM-NTF_{2} consisting of 1664 atoms, using the TZVP basis set resulting in 20192 functions.
Test Case B: Molecular Dynamics of liquid water, using 1024 water molecules. This case was adopted from the Preparatory Project. Some input keywords were altered to fit to the new branch of the code.
Test Case C: Molecular Dynamics of 64 OMIM-NTF_{2} using the TZV2P basis set. The system consists of 3328 atoms resulting in 54080 functions. Since the time needed to complete 1 MD step was too large, the last SCF iteration time was used to describe the performance.

3.2.3 *Results*

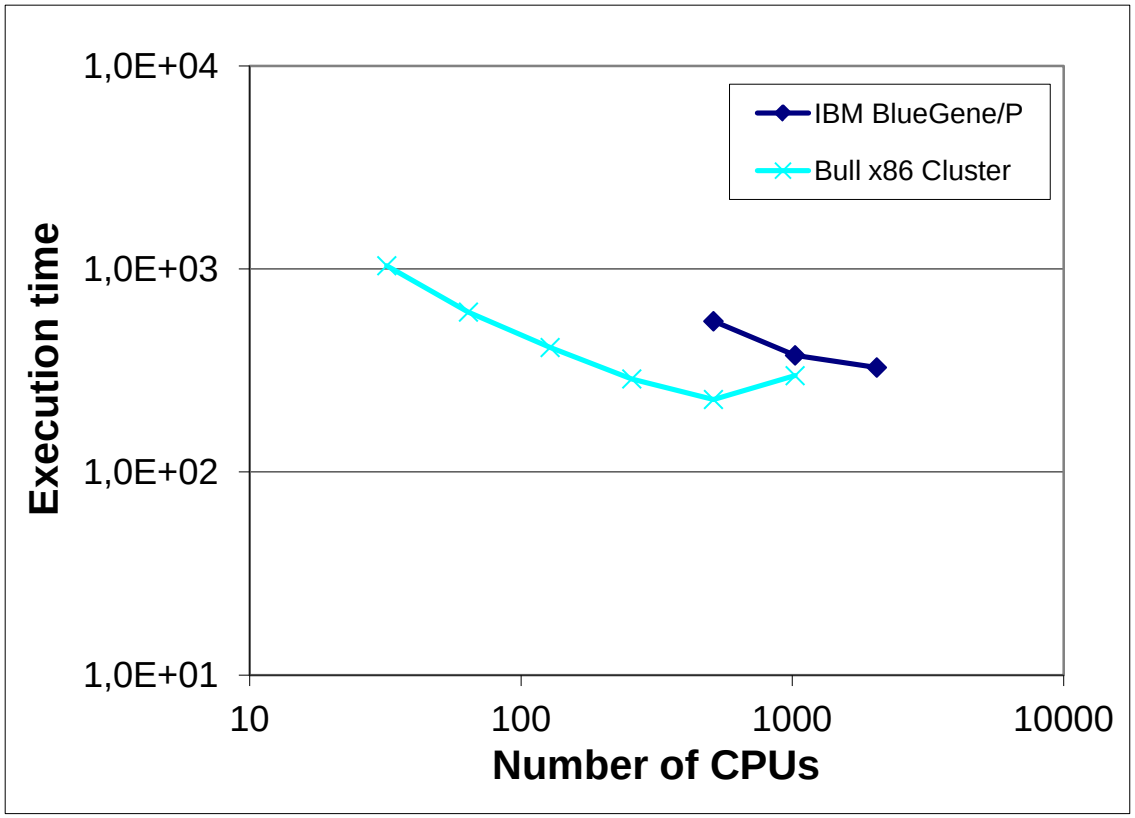

Figure 5: Execution time of CP2K, Test Case A

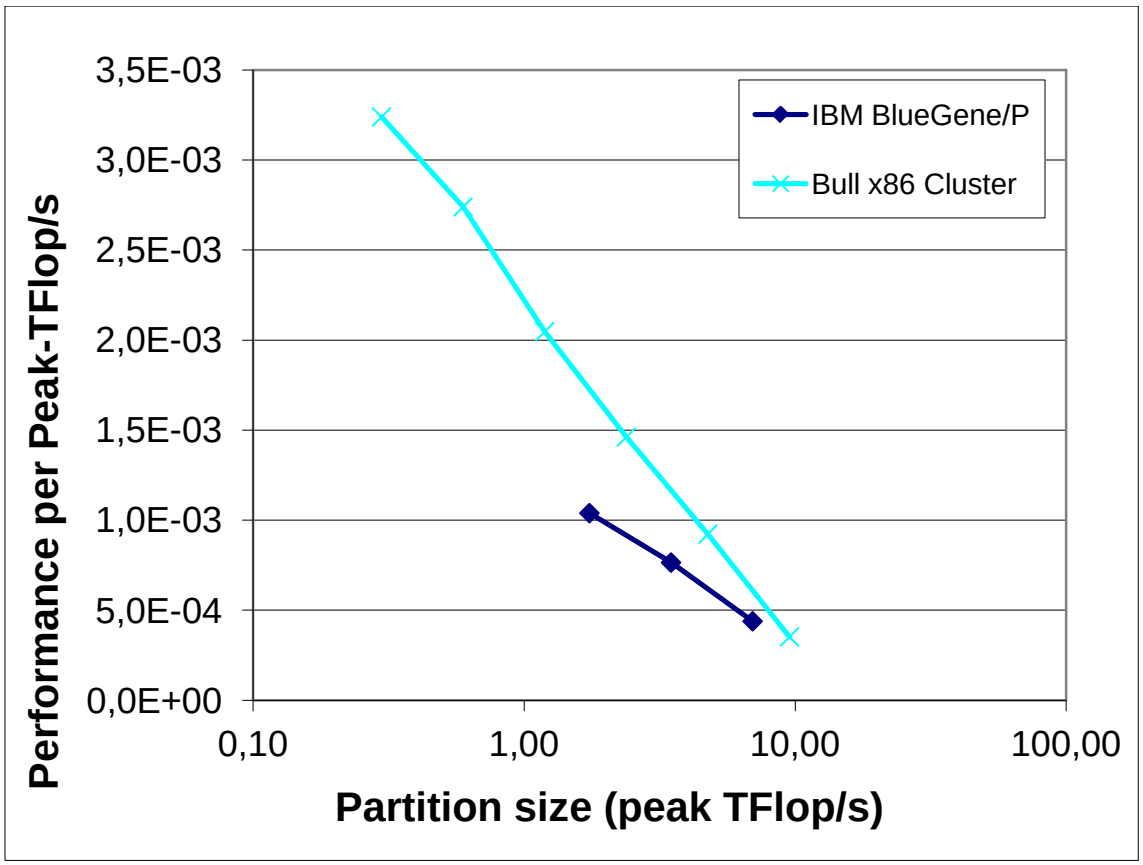

Figure 6: Performance per Peak-TFlop/s for CP2K, Test Case A

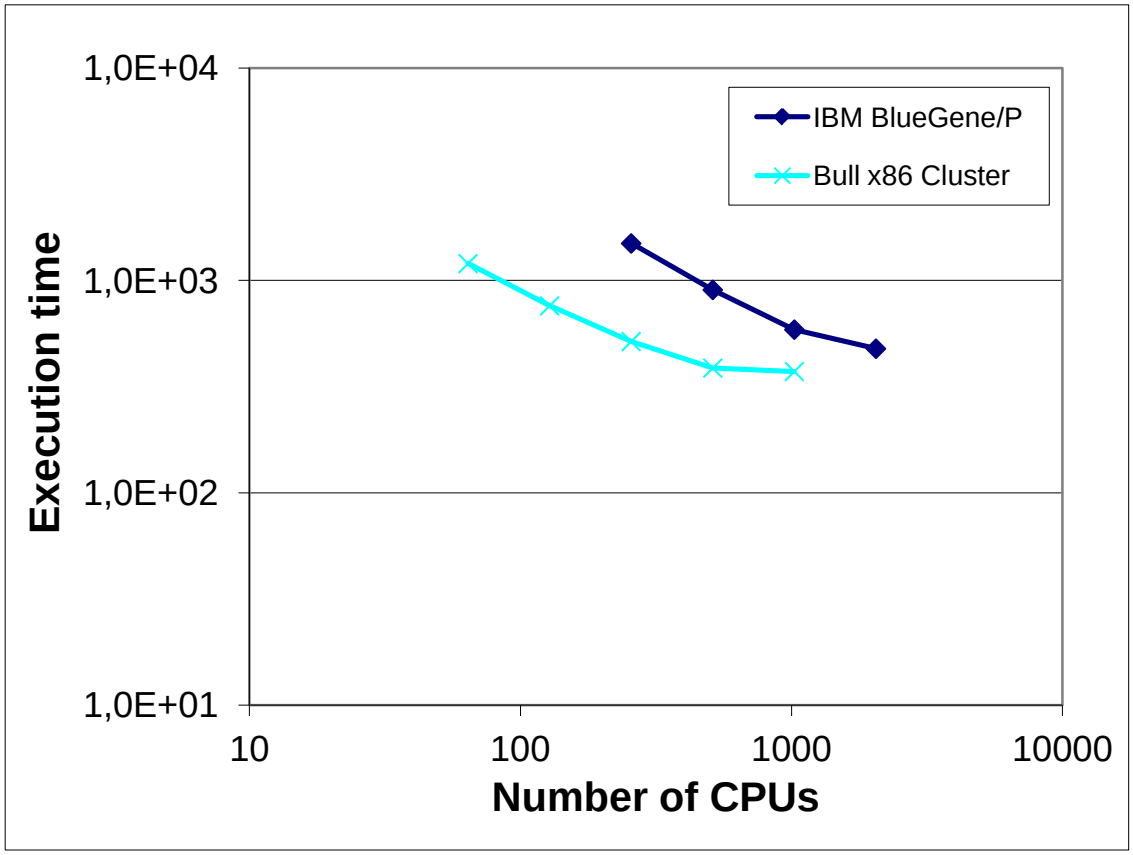

Figure 7: Execution time of CP2K, Test Case B

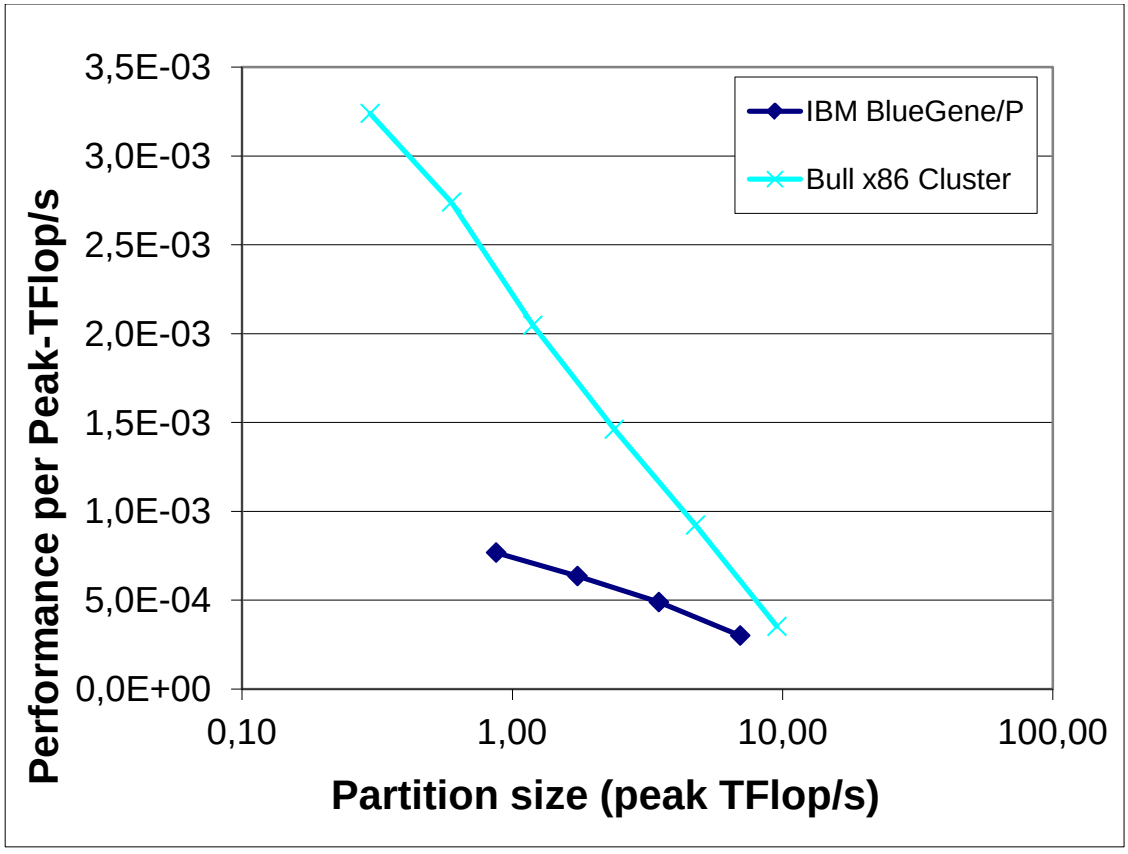

Figure 8: Performance per Peak-TFlop/s for CP2K, Test Case B

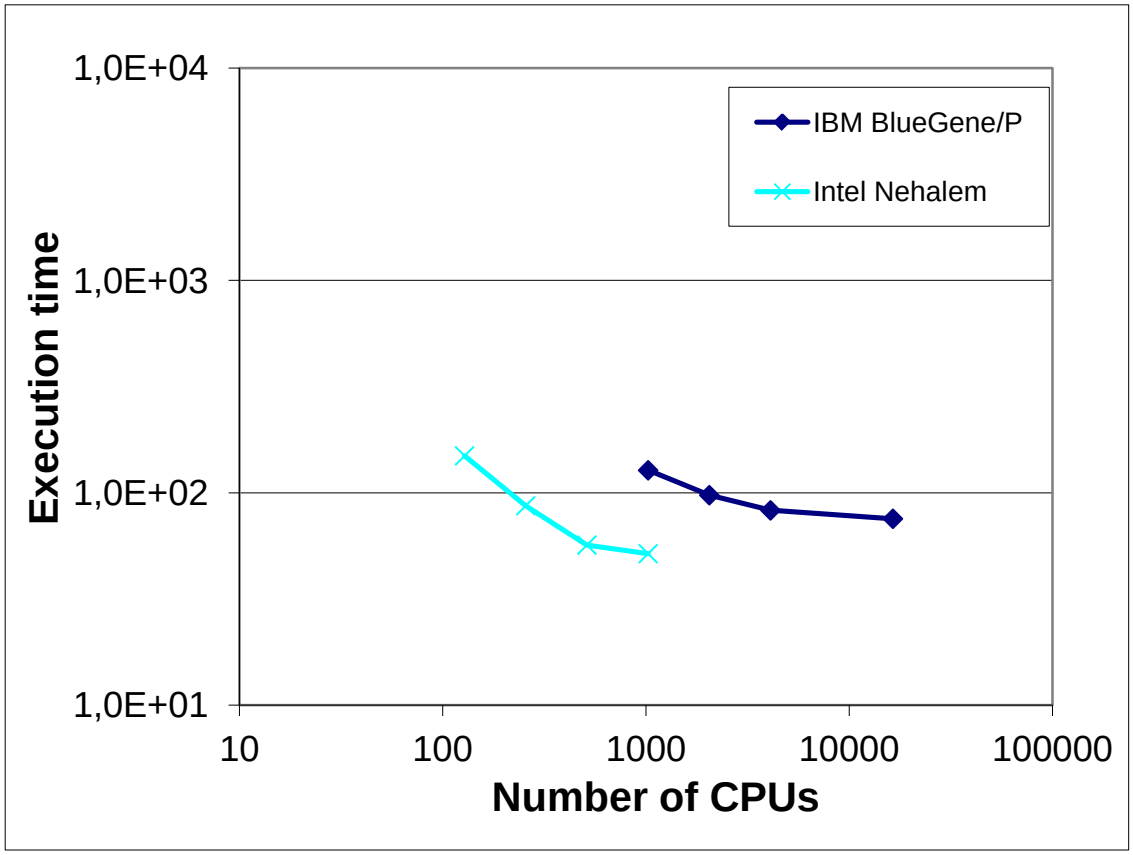

Figure 9: Execution time of CP2K, Test Case C

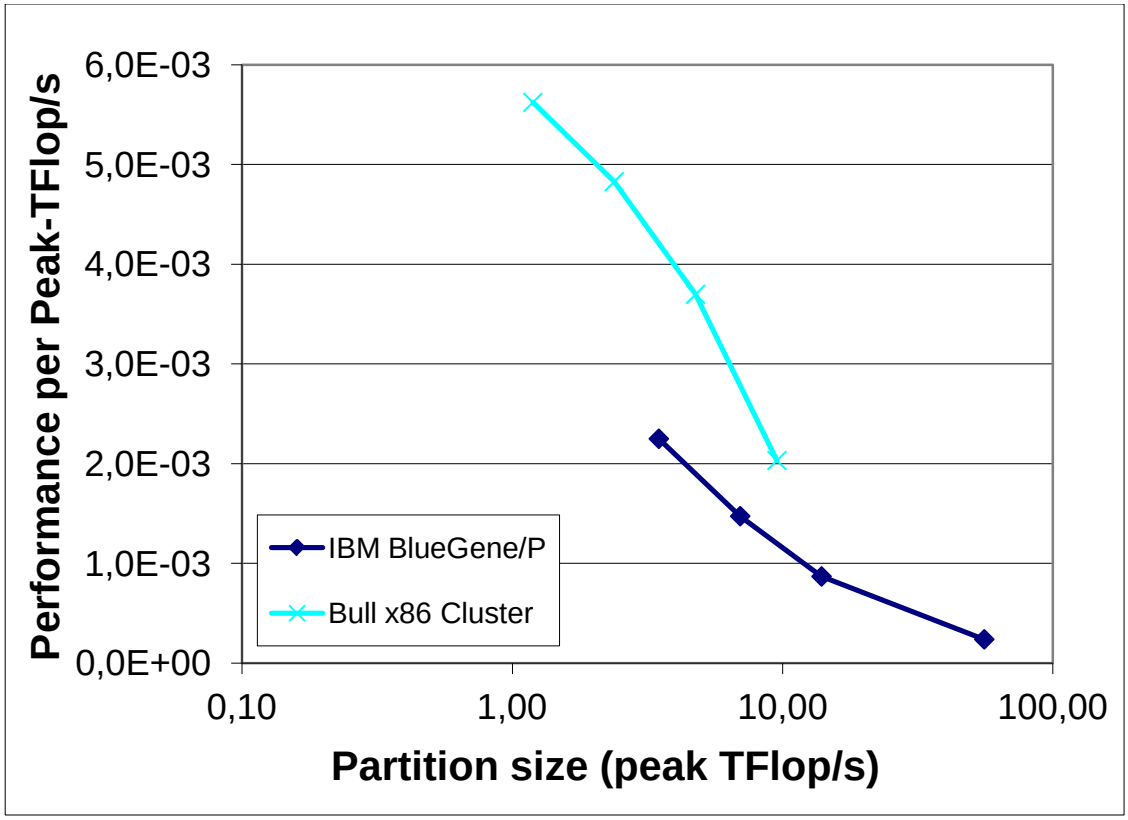

Figure 10: Performance per Peak-TFlop/s for CP2K, Test Case C

3.2.4 *Analysis*

A typical CP2K MD run includes:

1. wavefunction optimization - a relatively large number of SCF iterations.
2. a number of MD steps. Each MD step includes a number of SCF iterations

Step 1 is small for very small systems. As the system size increases step 1 takes more time to complete. In order to avoid repeating step 1 for all runs the wavefunction optimization was performed in one or more separate jobs and used subsequently as restart file for the benchmark runs.

A number of cases of increasing size/computation time requirements were prepared.

The biggest case that ran is that of test case C, for which we can get at least few SCF iterations within the benchmark job limit set to 30 min/job. Although it is common to report time per MD step, for this case we report the time for the last SCF iteration, since 30 min are not enough to complete a single MD step. Bigger cases require more than 30 min to complete the first SCF iteration.

Comparison of JUGENE and CURIE, for the same core count gives that CURIE is 2-3 times faster for the same case. CP2K shows relatively good speedup for the cases able to run up to 4096 on JUGENE and 1024 cores on CURIE, although runs slow down slightly up to 16384 cores on JUGENE.

3.3 CPMD

3.3.1 *Summary*

CPMD is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics.

3.3.2 *Test Cases*

Test Case A is a simulation of 64 Ionic Liquids.

Test Case B is a porphirin functionalized nanotube.

3.3.3 *Results*

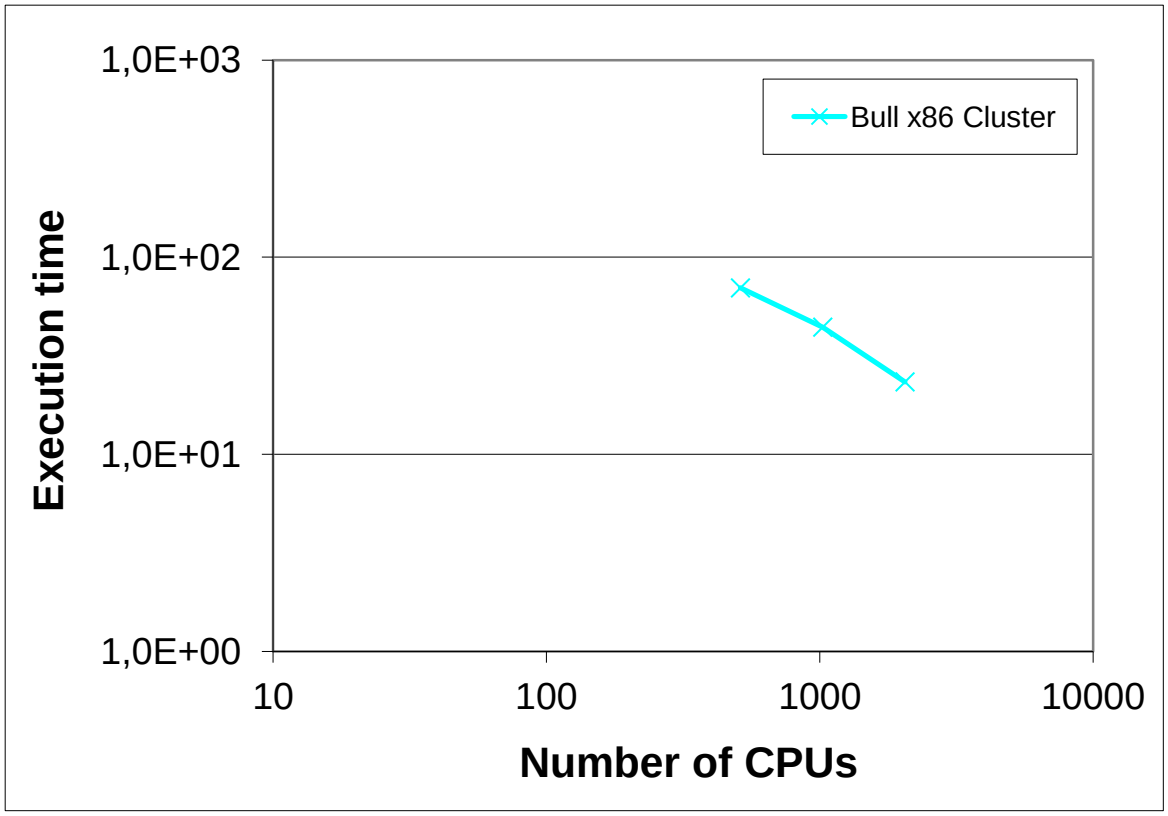

Figure 11: Execution time of CPMD, Test Case A

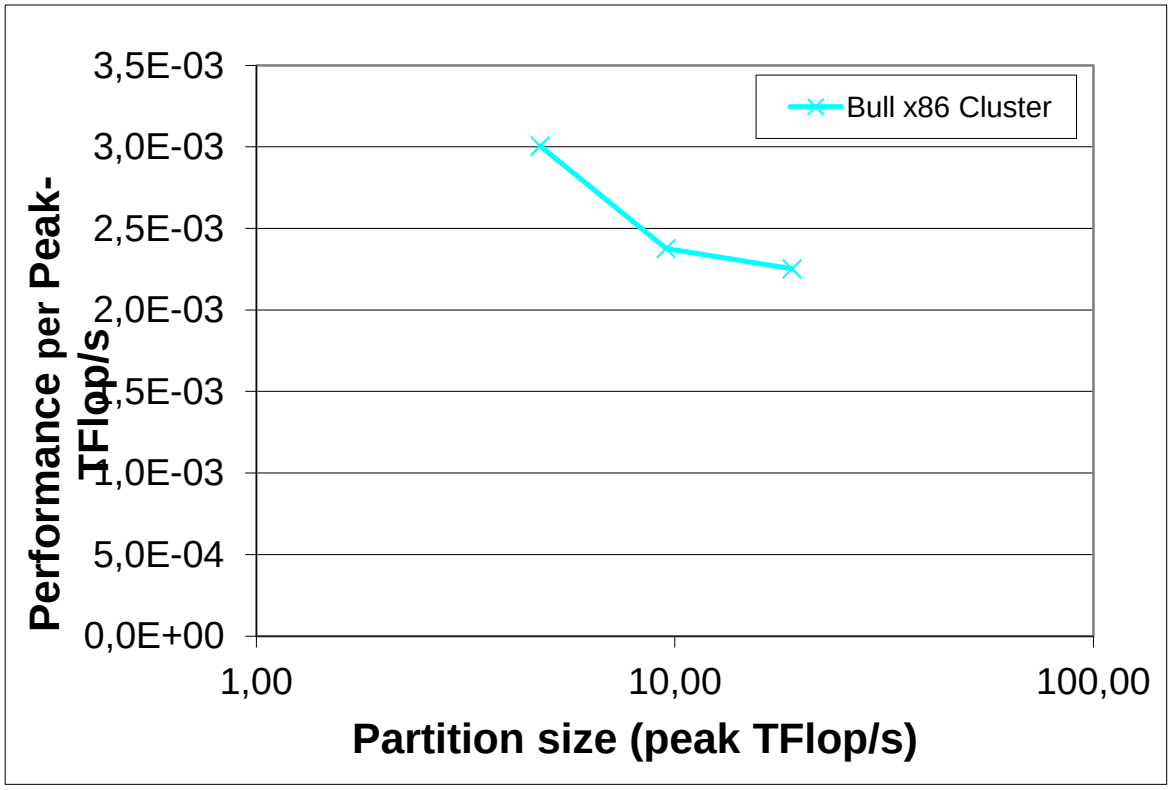

Figure 12: Performance per Peak-TFlop/s for CPMD, Test Case A

3.3.4 *Analysis*

The size of the test case is not large enough to scale to higher numbers of processors. Unfortunately, memory constraints do not allow the simulation of the larger test case (Test Case B), and even though the code should implement special data distribution functionality for large datasets, it was not possible to make it work properly, and simulations always crashed with memory allocation problems.

3.4 EUTERPE

3.4.1 *Summary*

EUTERPE is a particle-in-cell (PIC) gyrokinetic code for global linear and non-linear simulations of fusion plasma instabilities in three-dimensional geometries, in particular in tokamaks and stellarators. EUTERPE simulates a plasma in a cylindrical θ-pinch configuration. The code solves the coupled system of gyrokinetic equations for the ions, in the electrostatic approximation, and the quasi-neutrality equation, assuming adiabatically responding electrons.

3.4.2 *Test Cases*

Test Case A consists of a simulation of 1000 million particles.

Test Case B consists of a simulation of 8000 million particles.

3.4.3 *Results*

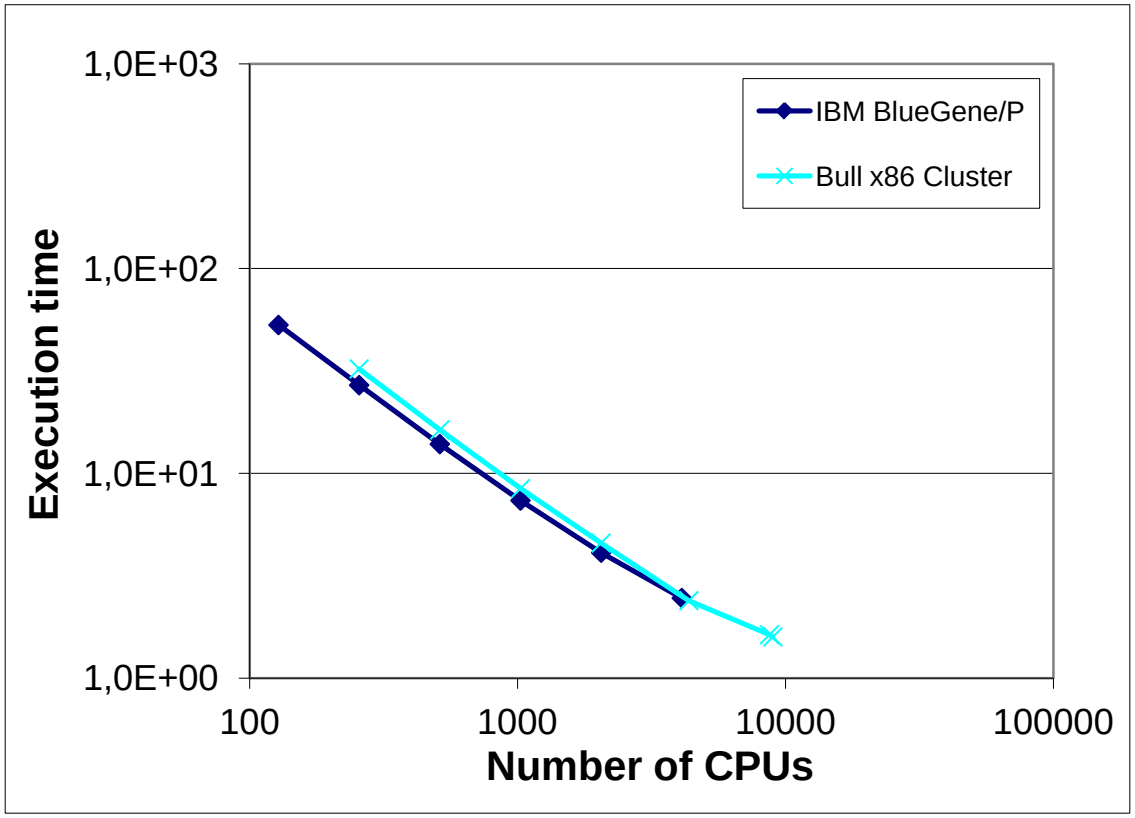

Figure 13: Execution time of EUTERPE, Test Case A

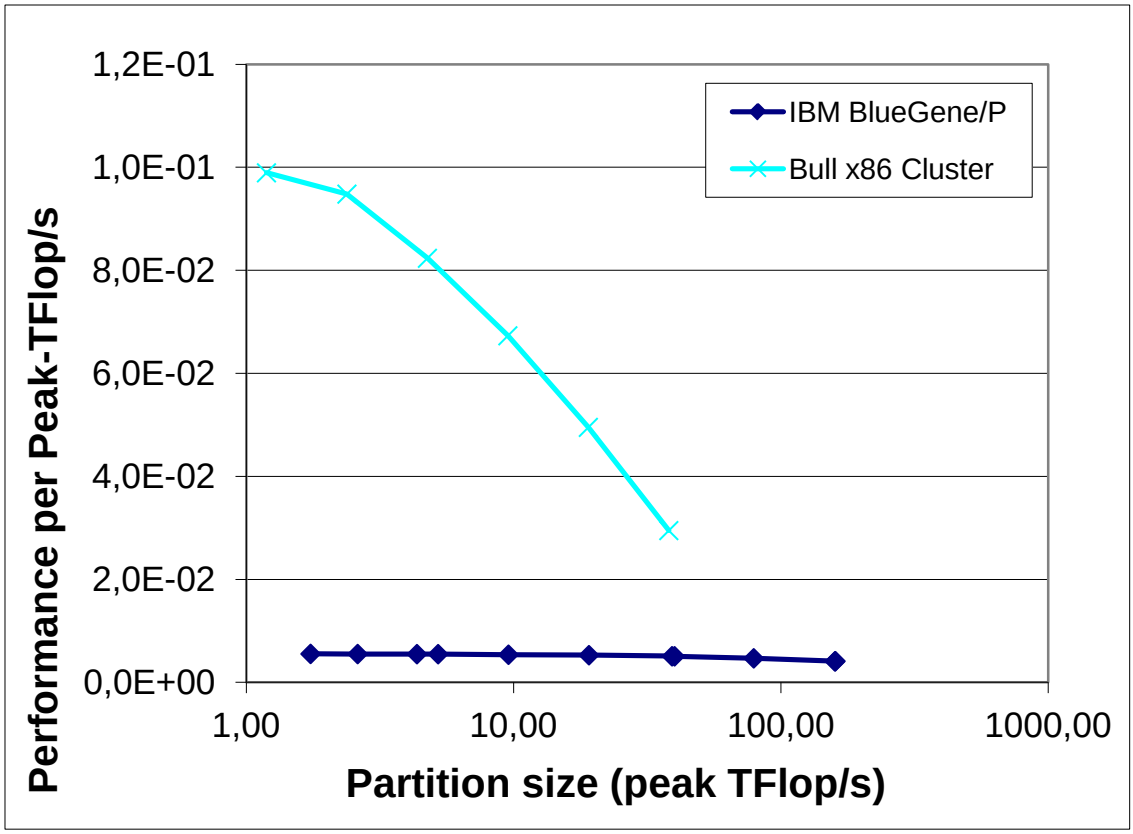

Figure 14: Performance per Peak-TFlop/s for EUTERPE, Test Case A

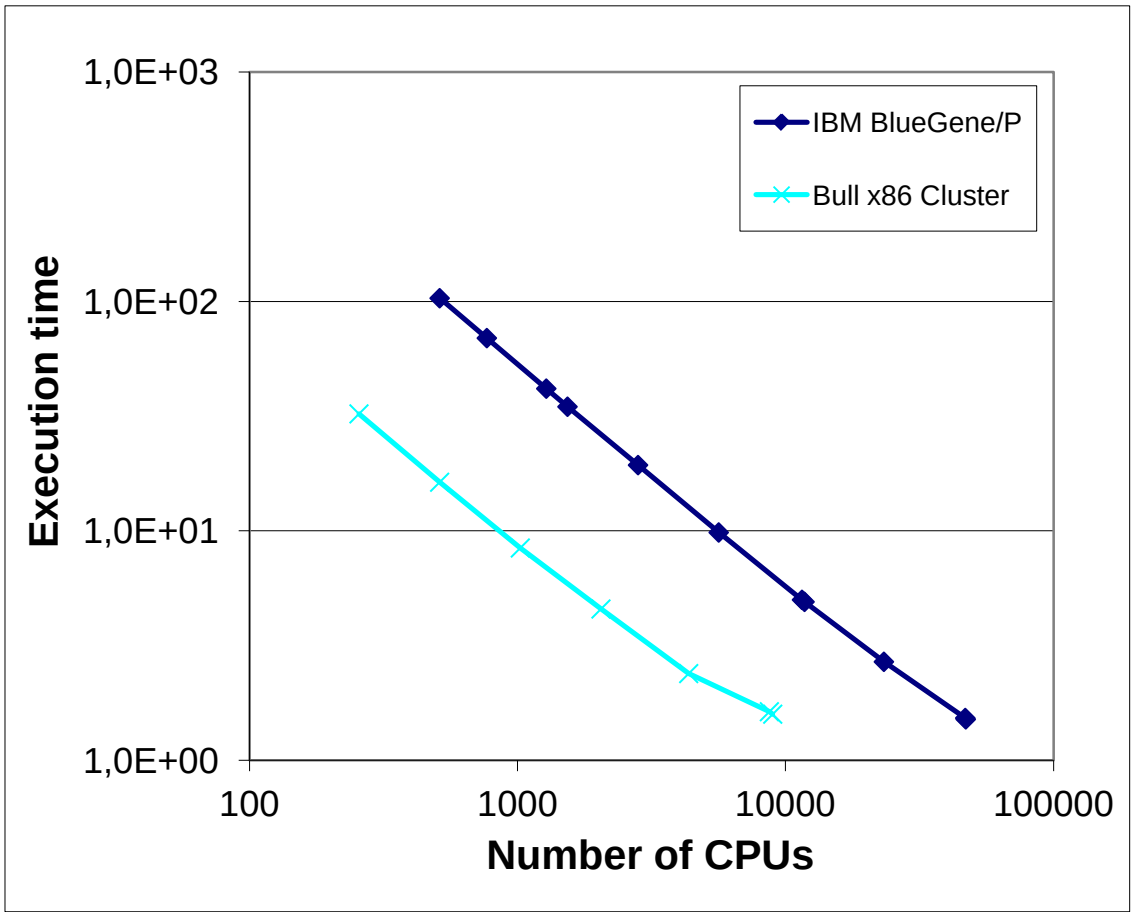

Figure 15: Execution time of EUTERPE, Test Case B

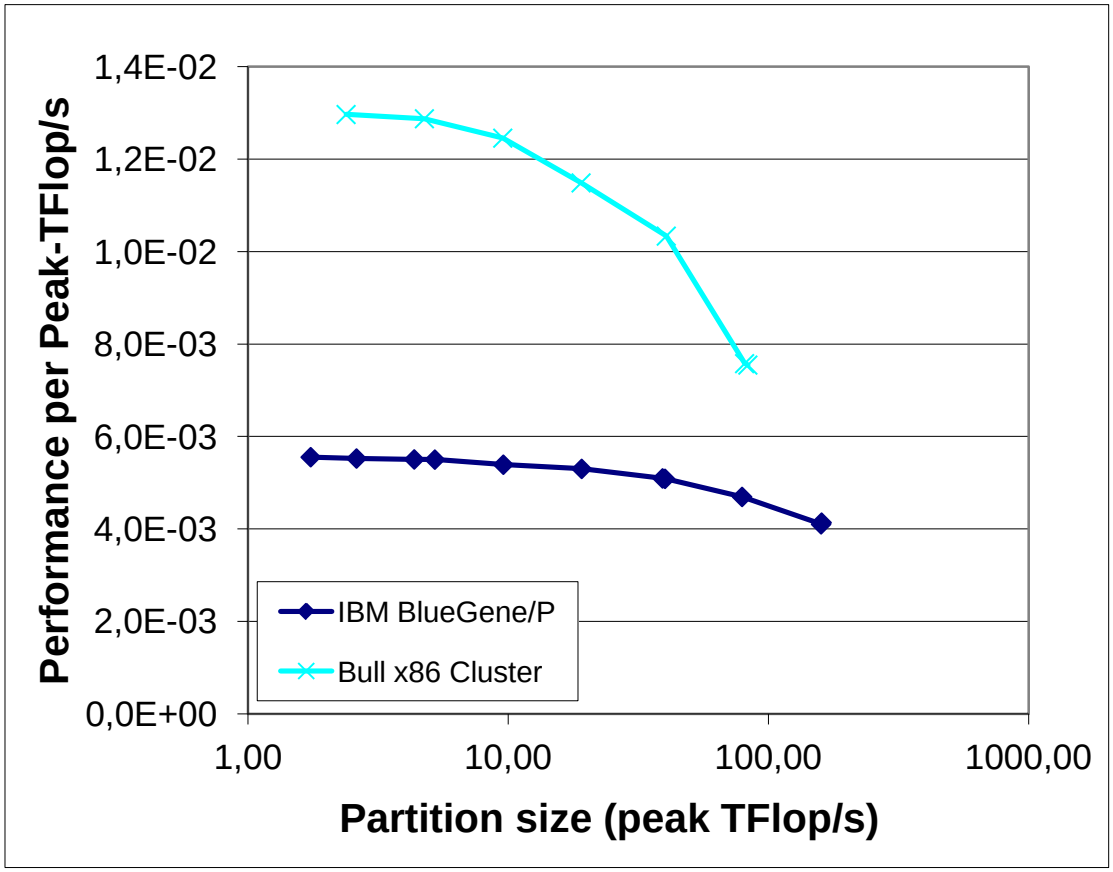

Figure 16: Performance per Peak-TFlop/s for EUTERPE, Test Case B

3.4.4 *Analysis*

The code has been parallelized using MPI. A domain cloning technique has been used to reduce the inter-processor communications. This strategy obtains very good performance up to a large number of processors. We have checked that the main limitations are the problem size to provide enough computational work to each processor and the reduced memory capacity of the IBM BlueGene/P nodes.

3.5 GADGET

3.5.1 *Summary*

GADGET is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory.

3.5.2 *Test Cases*

Test Case A consists of a simulation with 13824000 particles. Test Case A was used on the IBM Blue Gene/P.

Test Case B consists of a simulation with 32768000 particles. Test Case B was used on Bull x86 Cluster.

3.5.3 *Results*

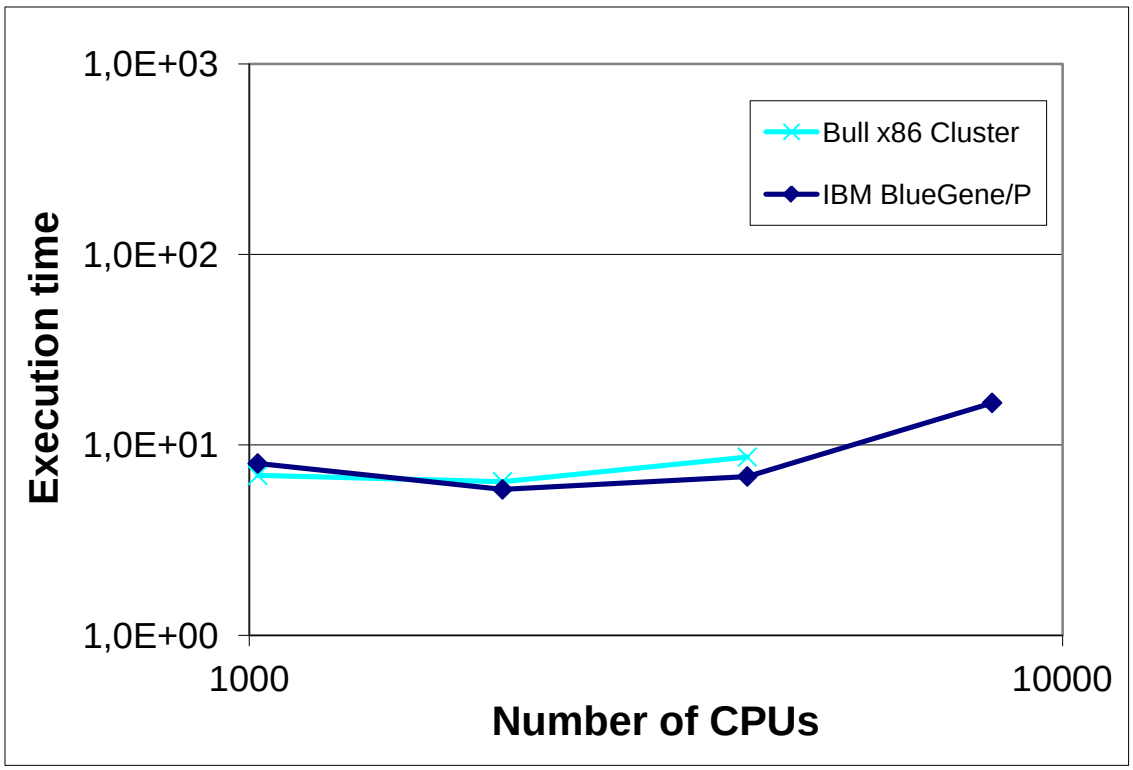

Figure 17: Execution time of GADGET, Test Case A for IBM BlueGene/P, Test Case B for Bull x86 Cluster

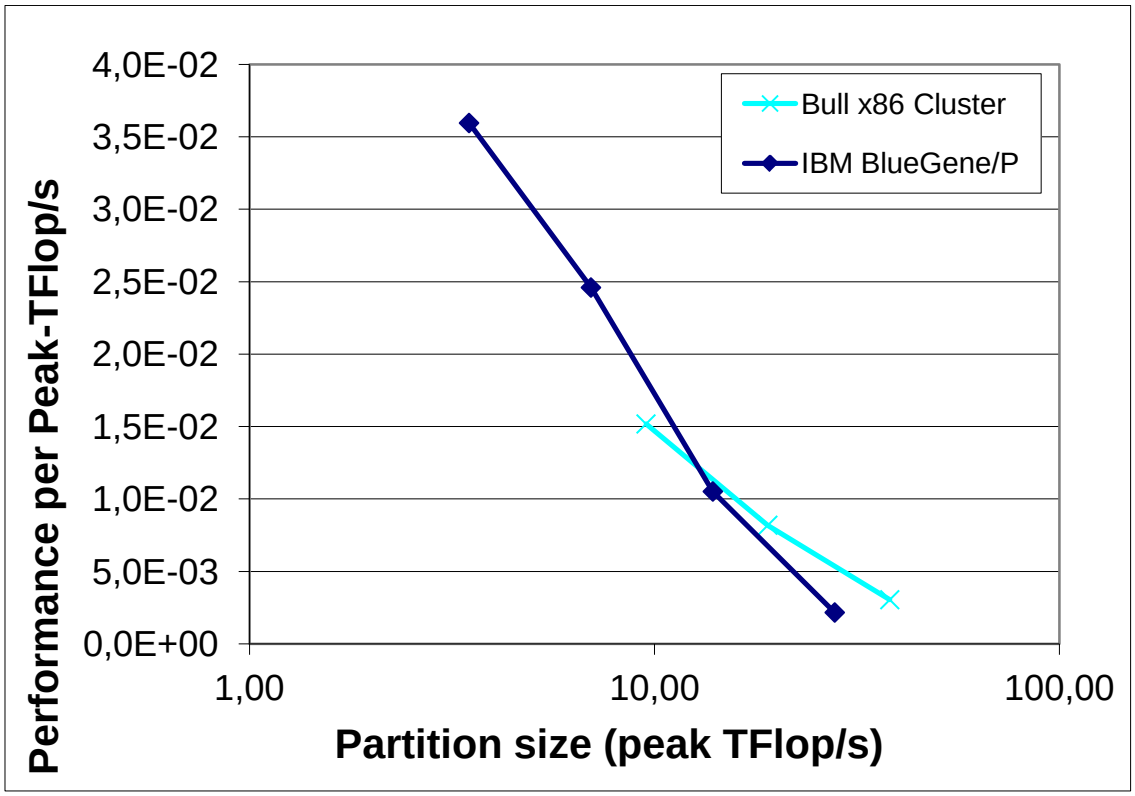

Figure 18: Performance per Peak-TFlop/s for GADGET, Test Case A for IBM BlueGene/P, Test Case B for Bull x86 Cluster

3.5.4 *Analysis*

Neither Test Case A nor Test Case B shows good scaling on IBM BlueGene/P or Bull x86 Cluster. There exists a new GADGET version 3, but this version is not freely available.

3.6 GROMACS

3.6.1 *Summary*

GROMACS is a versatile package to perform molecular dynamics, i.e. to simulate the Newtonian equations of motion for systems with hundreds to millions of particles. It is primarily designed for biochemical molecules such as proteins and lipids that have many complicated bonded interactions, but since GROMACS is extremely fast at calculating the non-bonded interactions (that usually dominate simulations) many groups are also using it for research on non-biological systems, e.g. polymers.

3.6.2 *Test Cases*

In all cases the number of PME nodes and domain decomposition was used which was auto-guessed by the code.

Test Case A: 1000 steps Molecular Dynamics of a system consisting of 125 MMPs in ethanol (1288227 atoms) using gromos96 forcefield and Particle Mesh Ewald method (PME) for electrostatics, heuristic update of neighbour lists.

Test Case B: 1000 steps Molecular Dynamics of 1 million SPC/E water molecules, PME for electrostatics.

3.6.3 *Results*

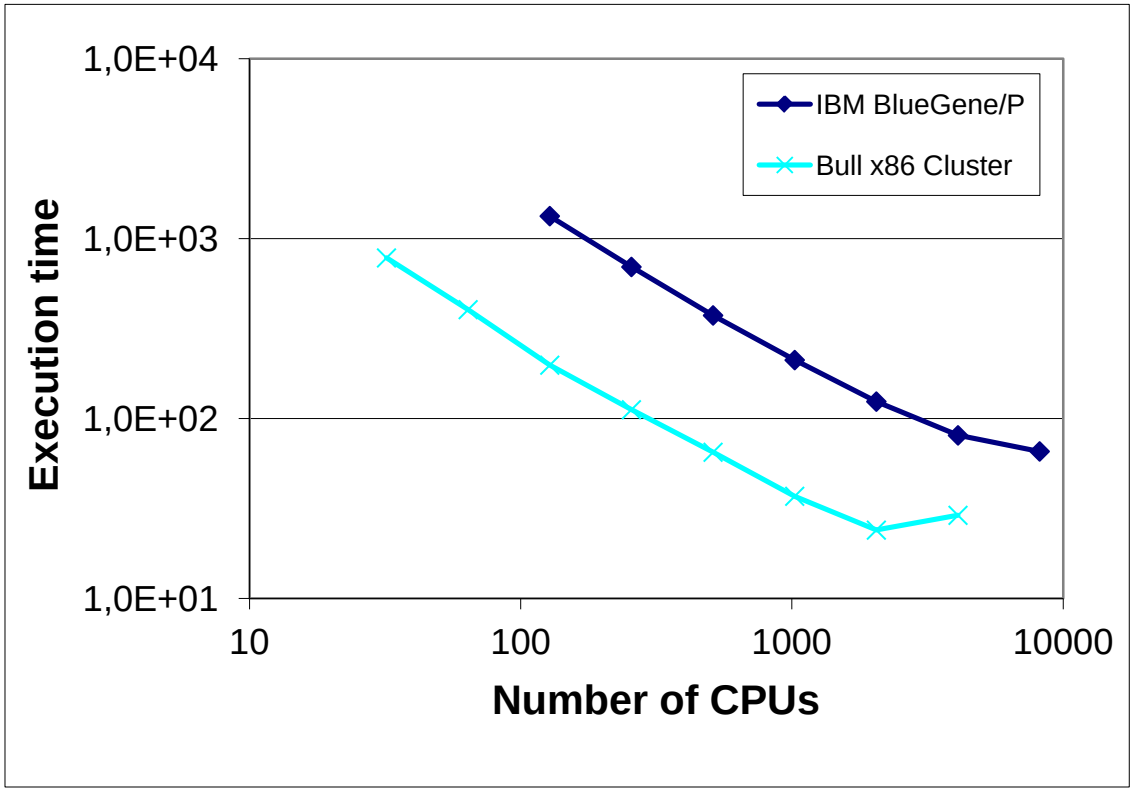

Figure 19: Execution time of GROMACS, Test Case A

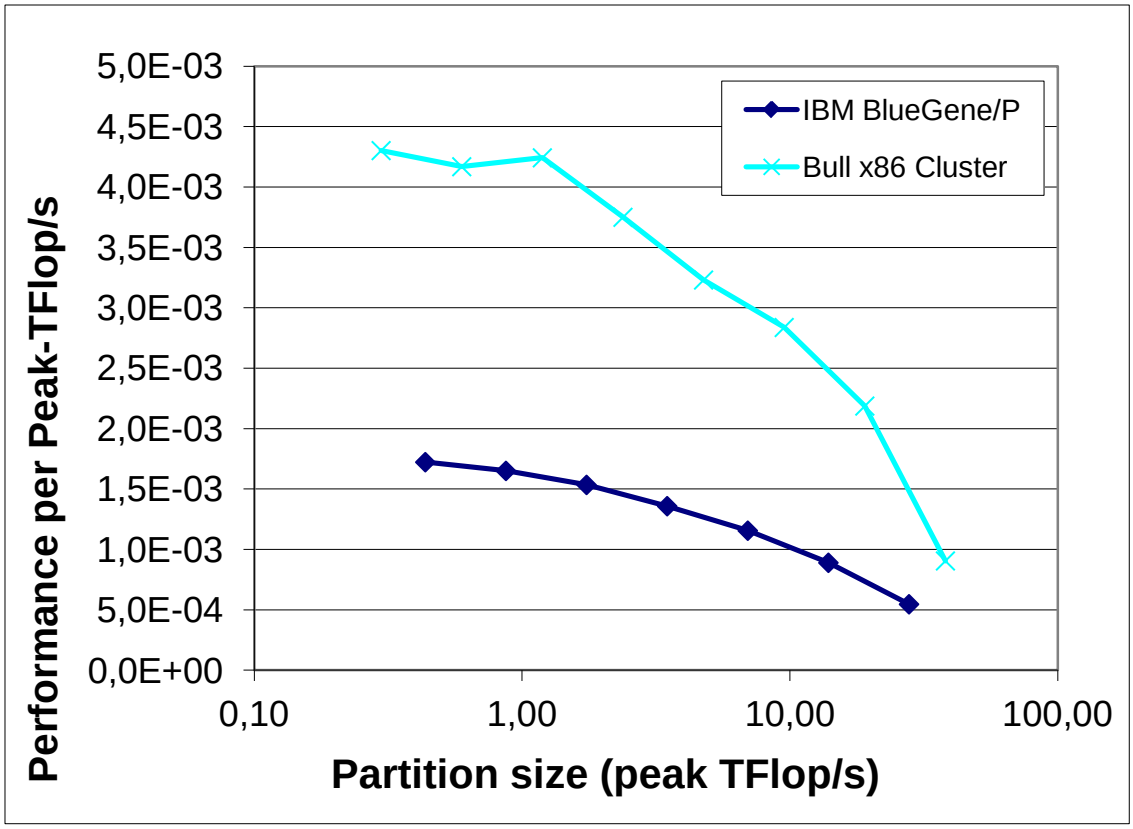

Figure 20: Performance per Peak-TFlop/s for GROMACS, Test Case A

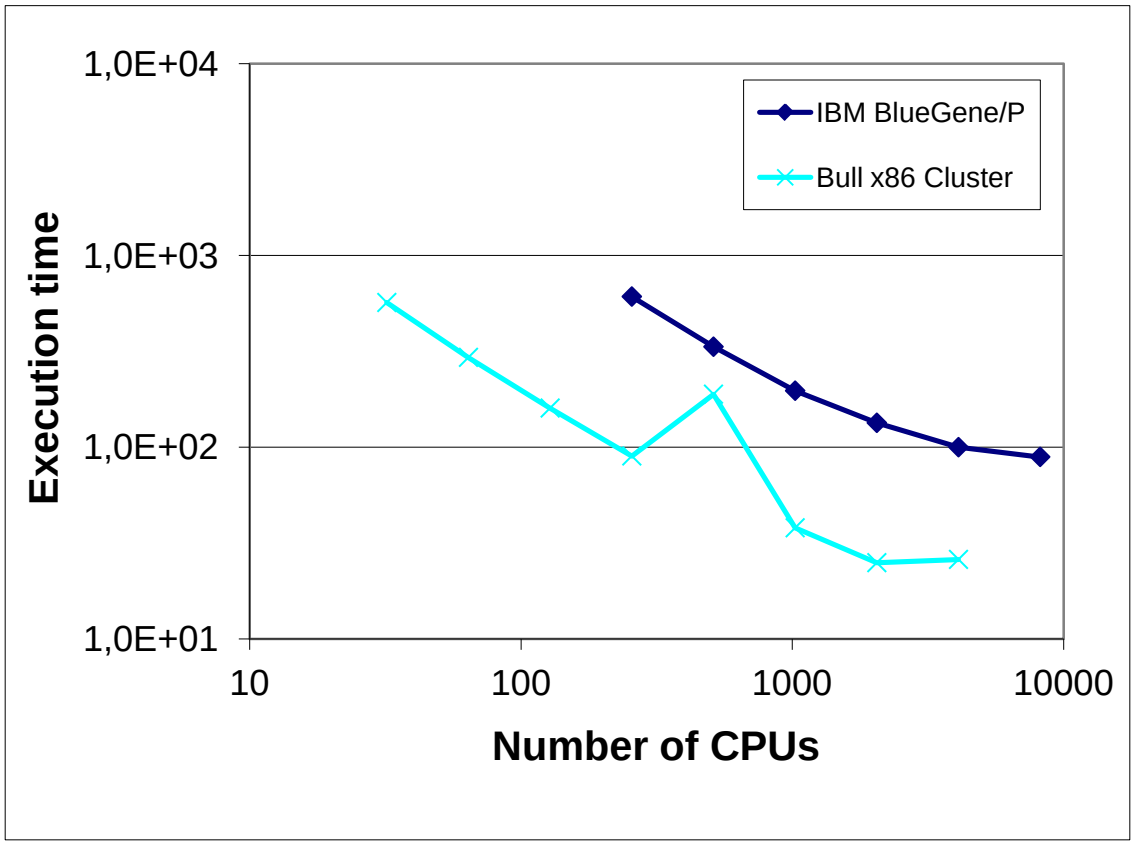

Figure 21: Execution time of GROMACS, Test Case B

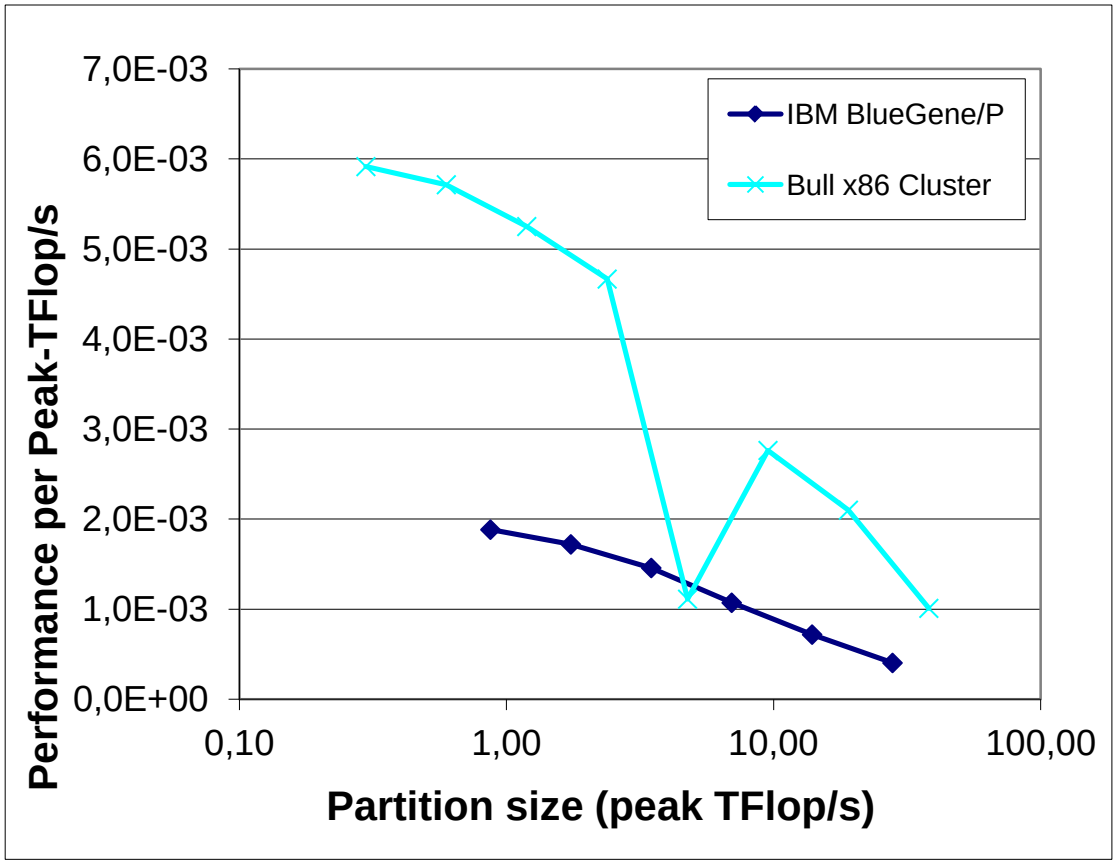

Figure 22: Performance per Peak-TFlop/s for GROMACS, Test Case B

3.6.4 *Analysis*

A number of parameters that affect GROMACS performance were tested, although only two large PME cases are reported here.

GROMACS exhibits relatively good scaling up to 4096 processors on JUGENE and 2048 processors on CURIE, although it runs with smaller efficiency up to 8192 and 4096 processors respectively.

A performance discontinuity appears on CURIE with 512 cores for the 3 million atoms case, this behaviour is reproducible. In all cases, the auto-guessed number of PME nodes was 1/4 of the total cores. In the case of 512 total cores on CURIE, the PME cores have more work to do than the rest of cores, leading to performance loss due to increased load imbalance.

The comparison of performance between JUGENE and CURIE shows that GROMACS is faster on CURIE than on JUGENE by a factor 4-5 for the same system and core count.

The GROMACS performance and scaling can be improved by further fine tuning of domain decomposition and PME on each system, core count and test case.

3.7 NAMD

3.7.1 *Summary*

NAMD is a parallel molecular dynamics code for high-performance simulation of large biomolecular systems.

3.7.2 *Test Cases*

Test Case A is a 1000 steps Molecular Dynamics of a TTR tetramer in water. The system contains 78113 atoms, electrostatics calculation every 2 steps.

Test Case B is a 1000 steps Molecular Dynamics of 6912 OMIM-NTF_{2} pairs. The system contains 359424 atoms, electrostatics calculation at every step.

Test Case C is a 1000 steps Molecular Dynamics of a protein in water. The system contains about 2.5 million atoms, electrostatics calculation every 5 steps. This case requires 2.2 GB of memory on the master process and was therefore only able to run on CURIE.

3.7.3 *Results*

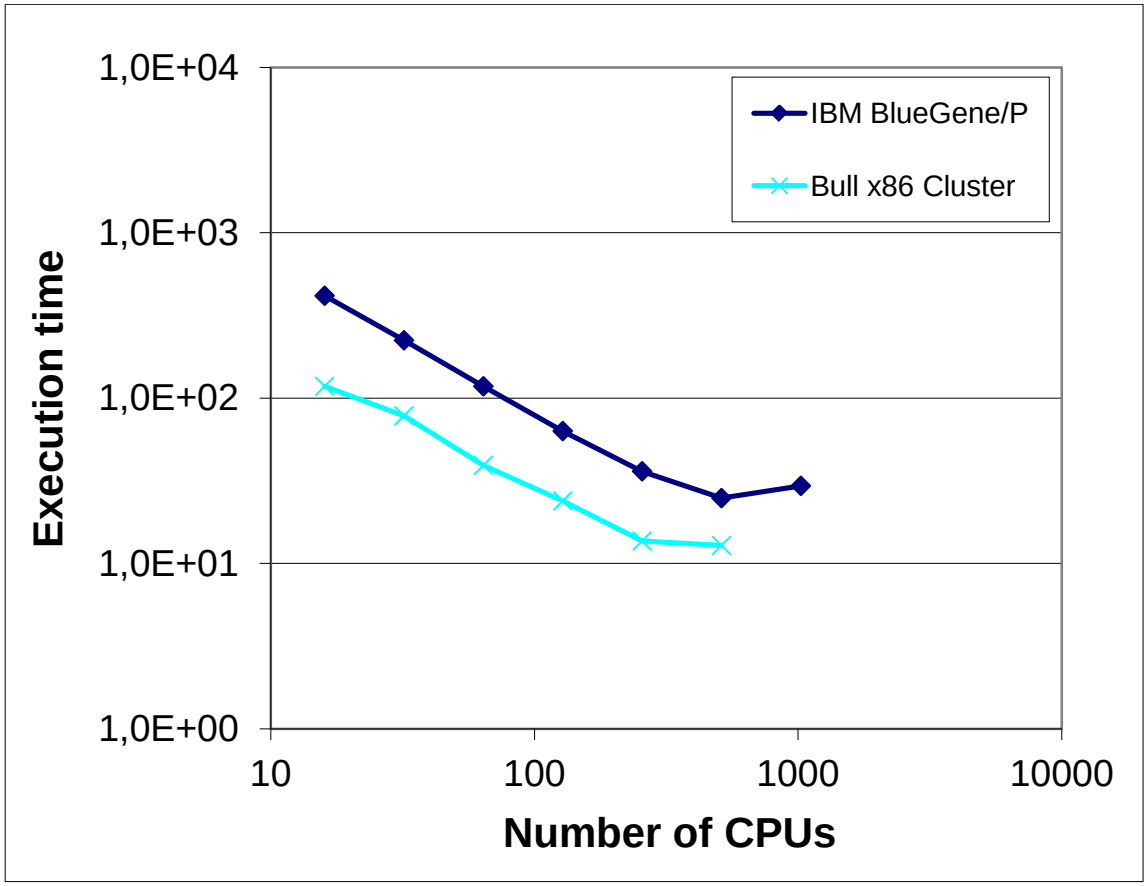

Figure 23: Execution time of NAMD, Test Case A

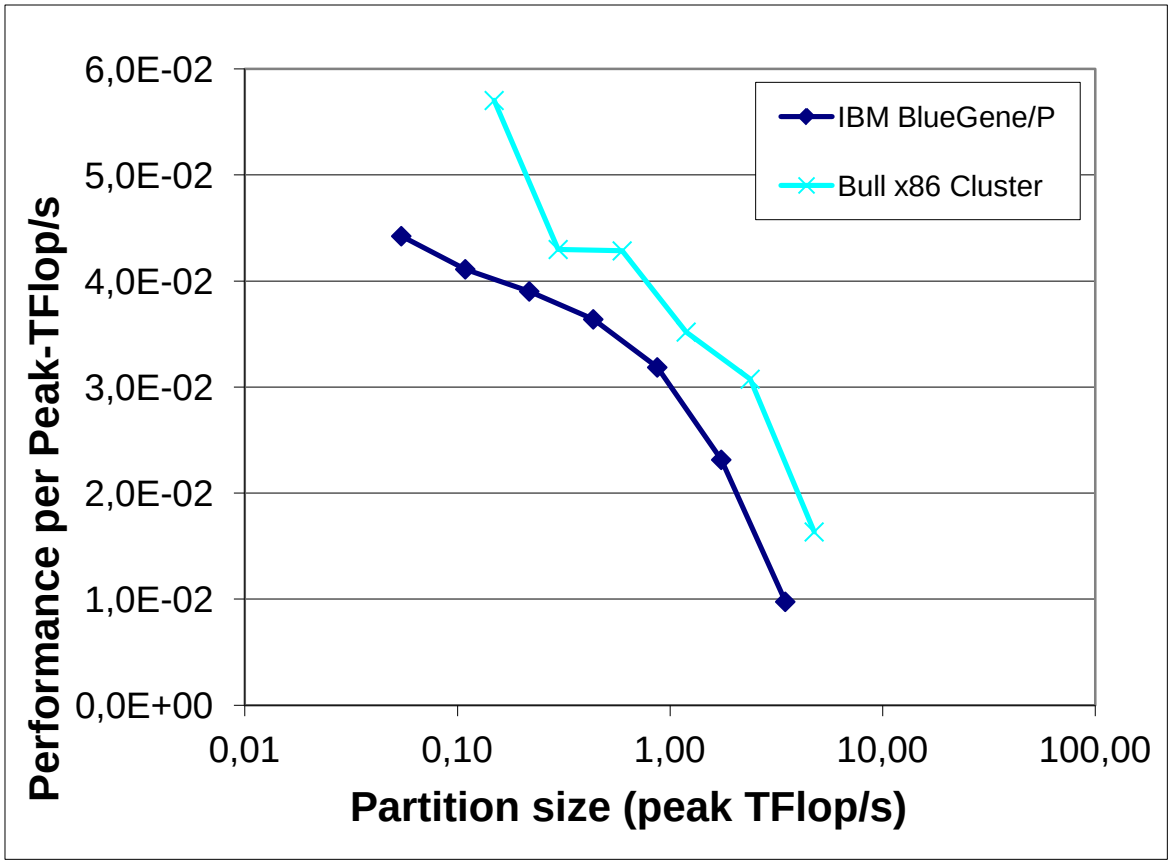

Figure 24: Performance per Peak-TFlop/s for NAMD, Test Case A

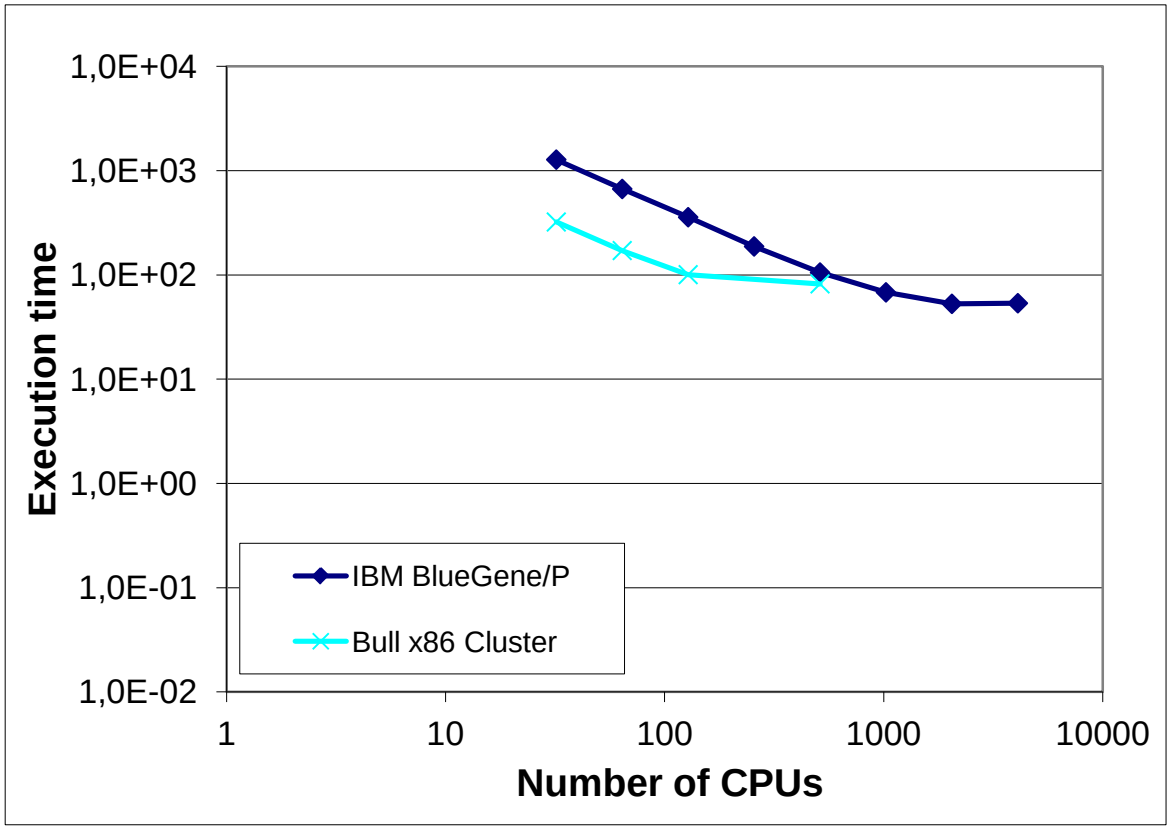

Figure 25: Execution time of NAMD, Test Case B

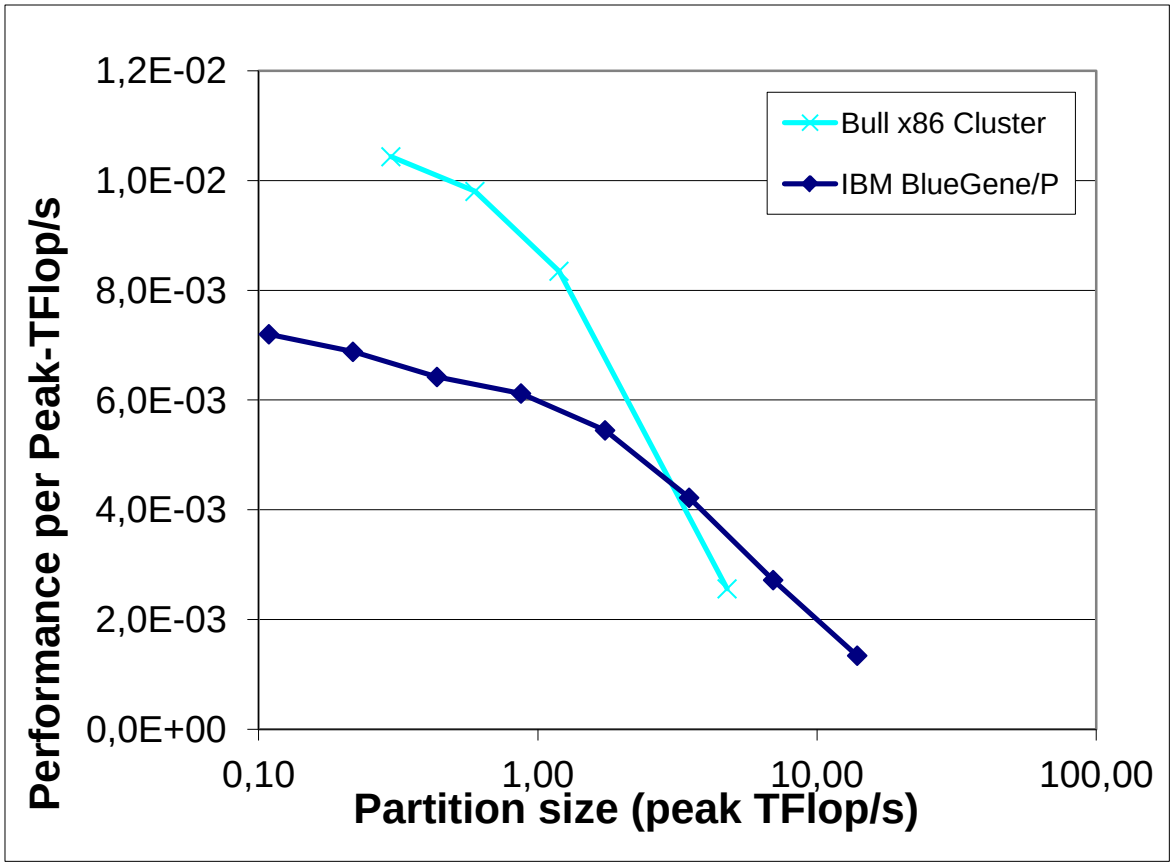

Figure 26: Performance per Peak-TFlop/s for NAMD, Test Case B

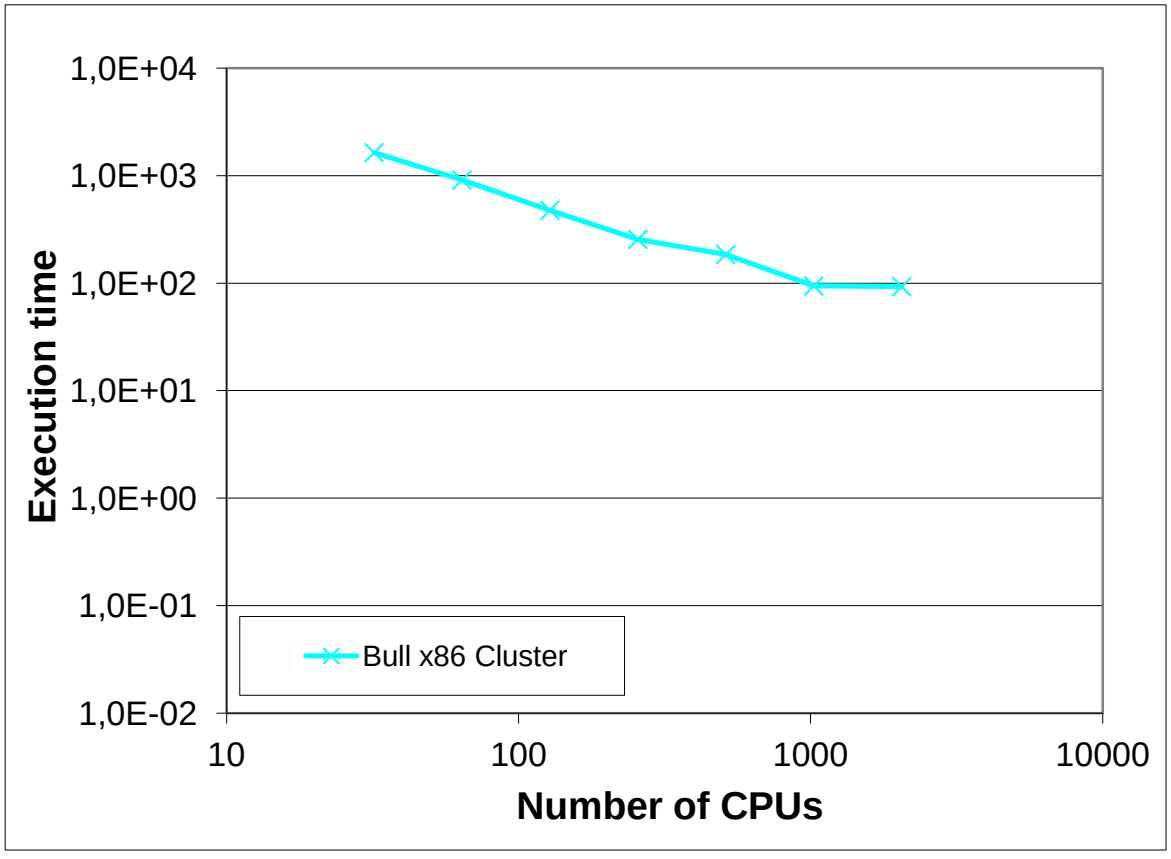

Figure 27: Execution time of NAMD, Test Case C

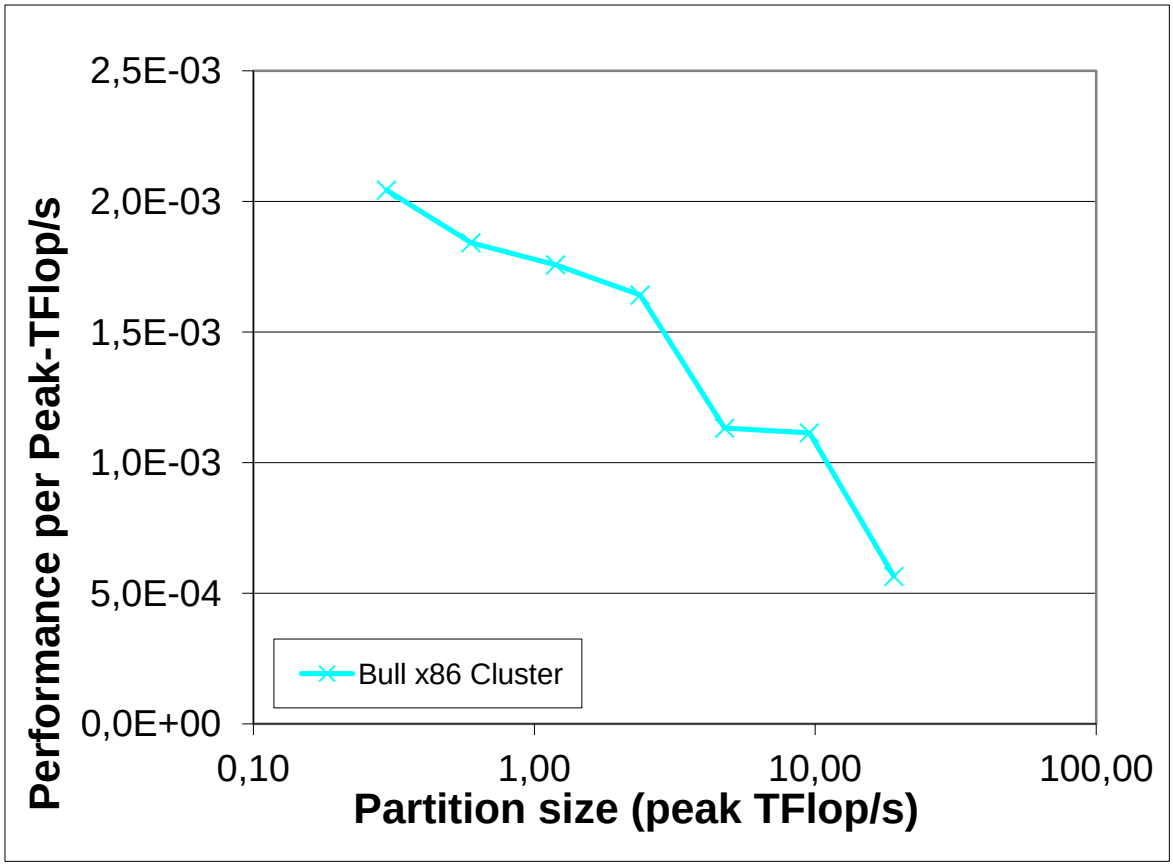

Figure 28: Performance per Peak-TFlop/s for NAMD, Test Case C

3.7.4 *Analysis*

For the NAMD benchmarks, the standard 2.8 version was used. We note that one can compile from the same source a low memory / high performance experimental version, but this version supports only a limited number of features. In the standard 2.8 version there are increased memory requirements on the master process, at least during startup. Thus, on JUGENE the case B was not able to run, it needs about 2.2 GB on the master process.

NAMD exhibits relatively good scaling up to 1024 cores.

All runs are performed for 1000 MD steps. NAMD has an internal Load Balancer that is updated during the run. In longer runs the load balancer is expected to optimize the load balancing between cores, in order to improve performance and scaling.

3.8 NEMO

3.8.1 *Summary*

NEMO is a numerical platform for simulating ocean dynamics, biochemistry, and sea-ice.

3.8.2 *Test Cases*

Test Case A is a simulation of the global ocean with sea-ice at a resolution of 1/12th degree.

3.8.3 *Results*

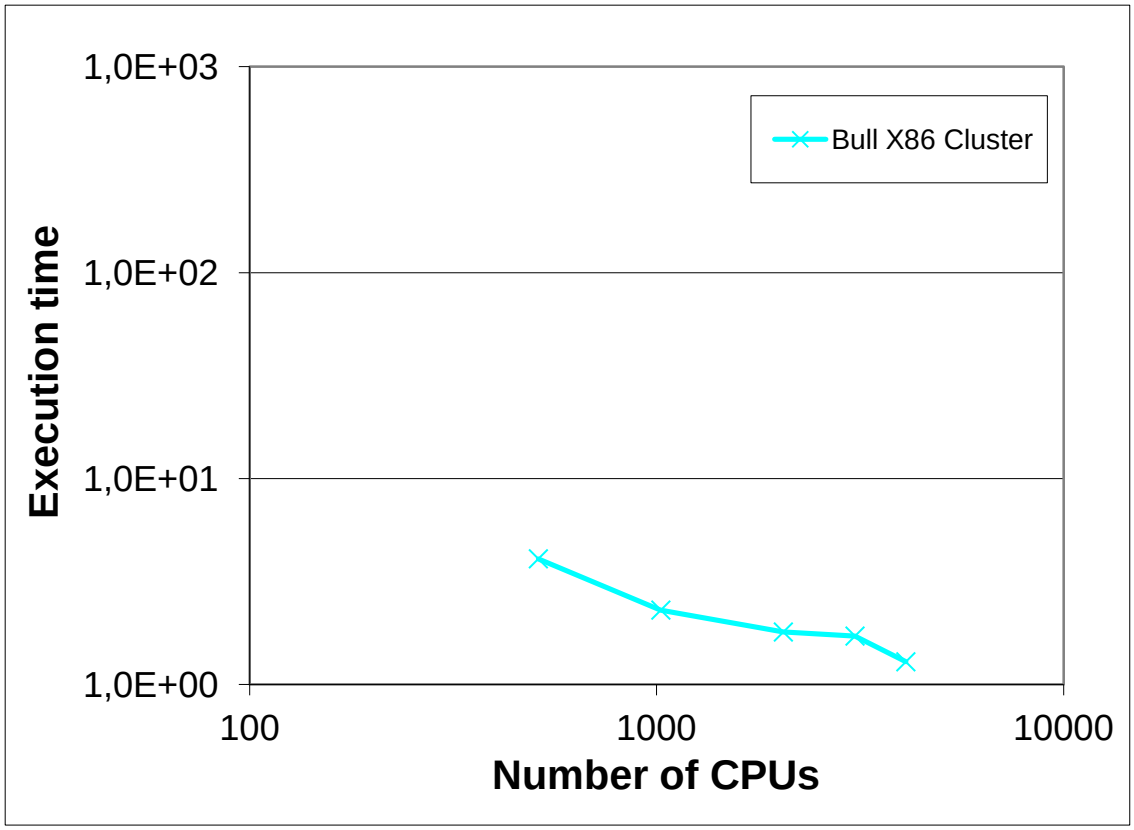

Figure 29: Execution time of NEMO, Test Case A

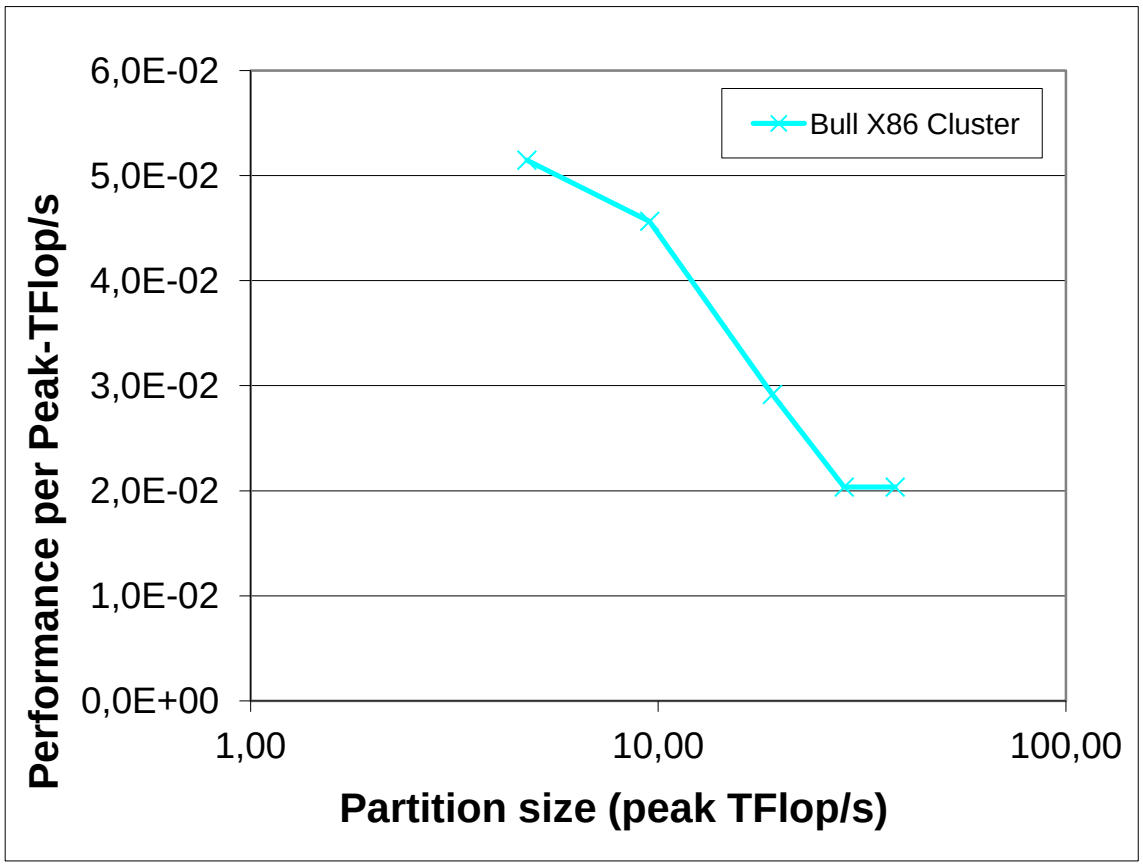

Figure 30: Performance per Peak-TFlop/s for NEMO, Test Case A

3.8.4 *Analysis*

The simulation measures 100 time steps, excluding the first 5 time steps. During all time steps the ocean evolution is computed, every 6th timestep also the sea-ice evolution. One timestep writes the output. At the highest scales, the I/O timestep on CURIE can take up to 30% of the runtime. A new I/O server is under development. Furthermore, MPI flags to optimize communication time for NEMO on CURIE were not yet used in these computations.

3.9 NS3D

3.9.1 *Summary*

NS3D solves the incompressible Navier-Stokes equations by Direct Numerical Simulation (DNS).

3.9.2 *Test Cases*

Test Case A is a shearwake with 1024x840x128 grid points with one time step computed.

3.9.3 *Results*

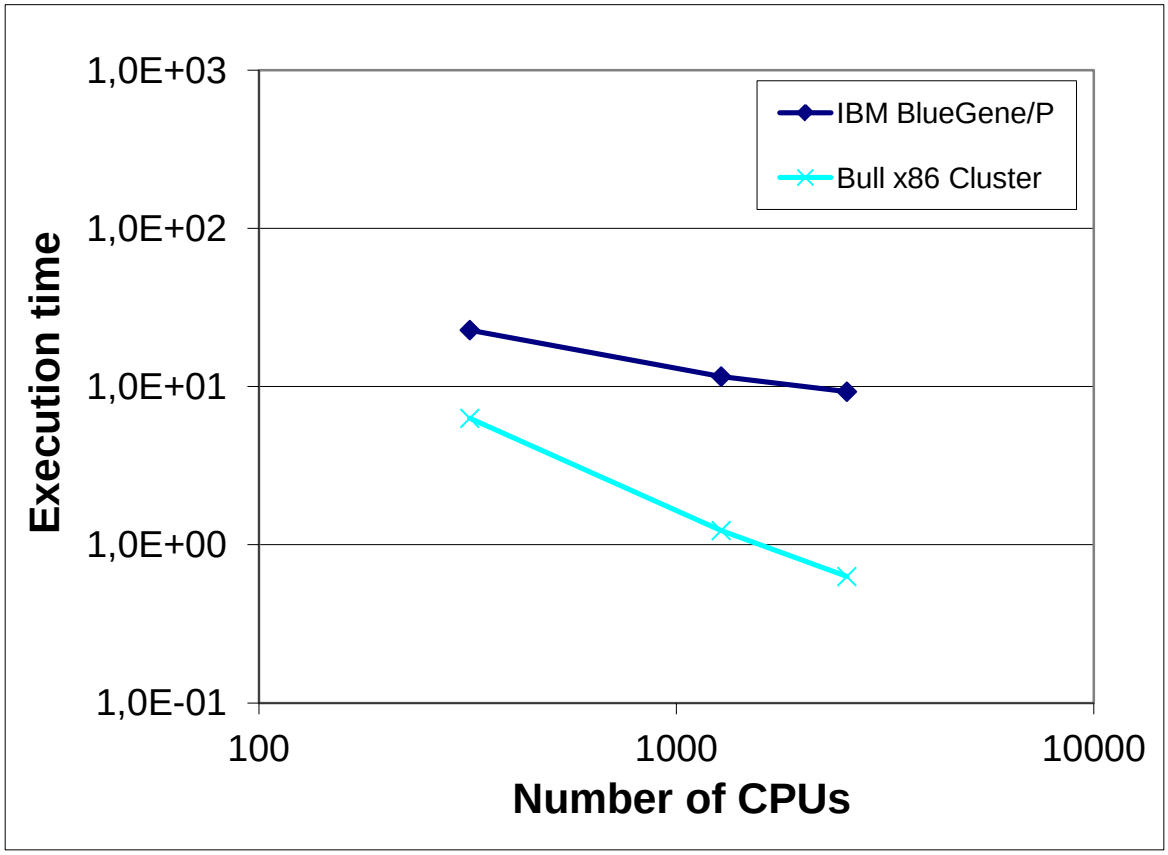

Figure 31: Execution time of NS3D, Test Case A

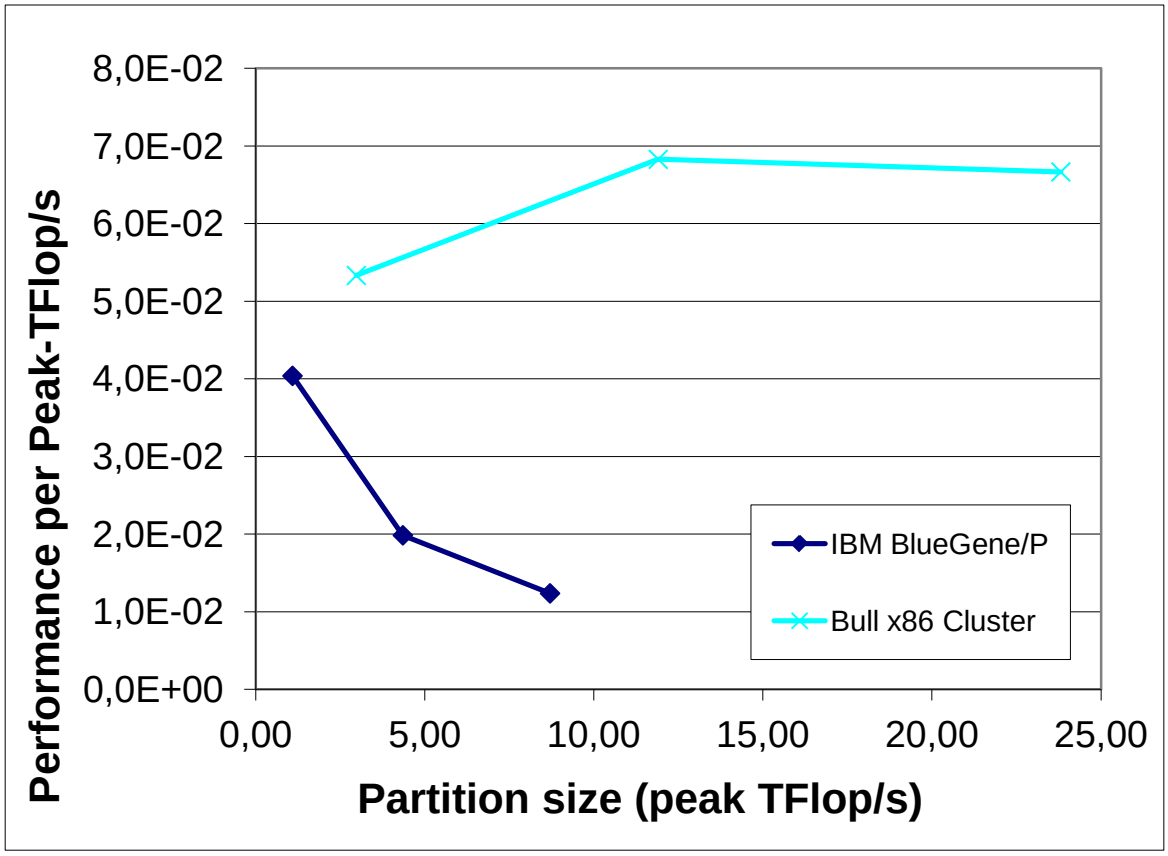

Figure 32: Performance per Peak-TFlop/s for NS3D, Test Case A

3.9.4 *Analysis*

The code scales quite well up to 23TFlop/s. The scaling behaviour is slightly better on the Bull x86 Cluster than on the Blue Gene/P.

3.10 QCD

3.10.1 *Summary*

The QCD benchmark is, unlike the other benchmarks in the PRACE application benchmark suite, not a full application but a set of 5 kernels which are representative of some of the most compute-intensive parts of QCD calculations.

3.10.2 *Test Cases*

Each of the 5 kernels has one test case:

Kernel A is derived from BQCD (Berlin Quantum ChromoDynamics program), a hybrid Monte-Carlo code that simulates Quantum Chromodynamics with dynamical standard Wilson fermions. The computations take place on a four-dimensional regular grid with periodic boundary conditions. The kernel is a standard conjugate gradient solver with even/odd pre-conditioning. Lattice size is $32^2 \times 64^2$.

Kernel B is derived from SU3_AHiggs, a lattice quantum chromodynamics (QCD) code intended for computing the conditions of the Early Universe. Instead of "full QCD", the code applies an effective field theory, which is valid at high temperatures. In the effective theory, the lattice is 3D. Lattice size is 256^3.

Kernel C Lattice size is 8^4. Note that Kernel C can only be run in a weak scaling mode, where each CPU stores the same local lattice size, regardless of the number of CPUs. Ideal scaling for this kernel therefore corresponds to constant execution time, and performance per peak TFlop/s is simply the reciprocal of the execution time.

Kernel D consists of the core matrix-vector multiplication routine for standard Wilson fermions. The lattice size is 64^4 .

Kernel E consists of a full conjugate gradient solution using Wilson fermions. Lattice size is 64^3 x 3.

3.10.3 *Results*

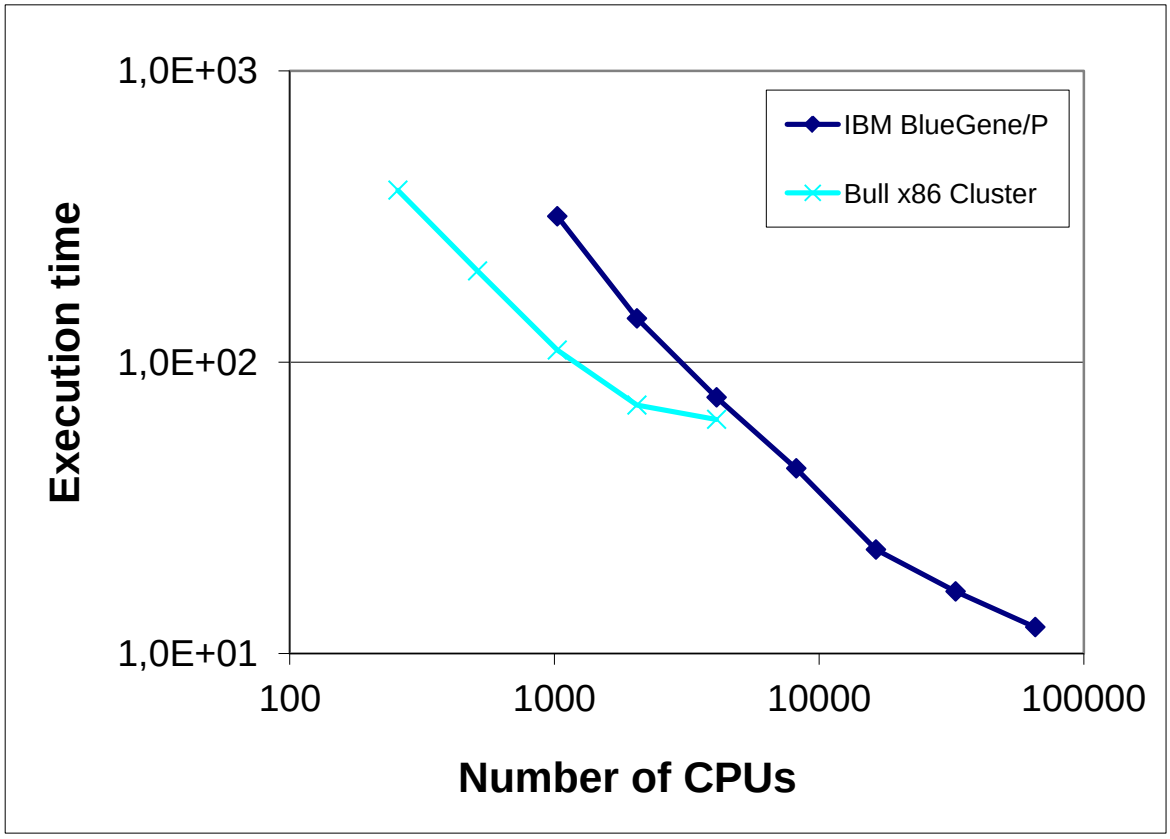

Figure 33: Execution time of QCD, Kernel A

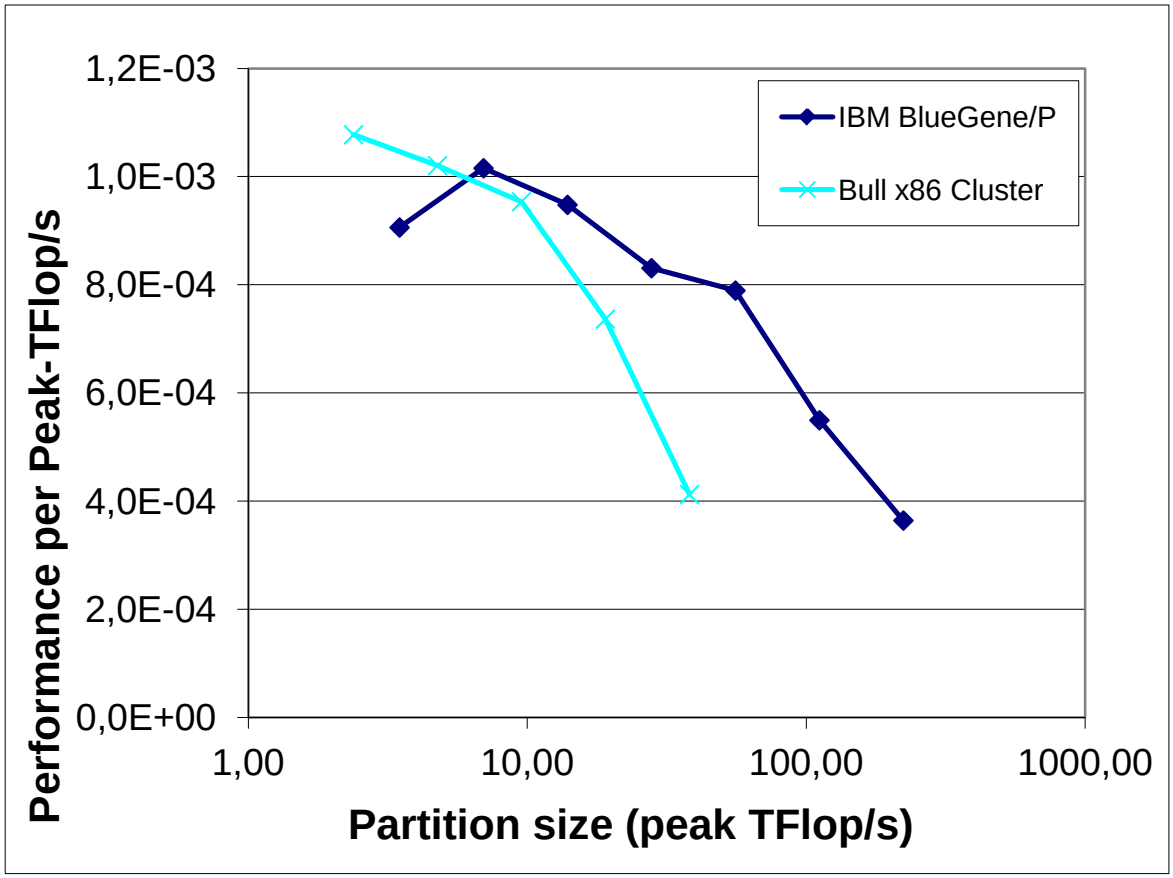

Figure 34: Performance per Peak-TFlop/s for QCD, Kernel A

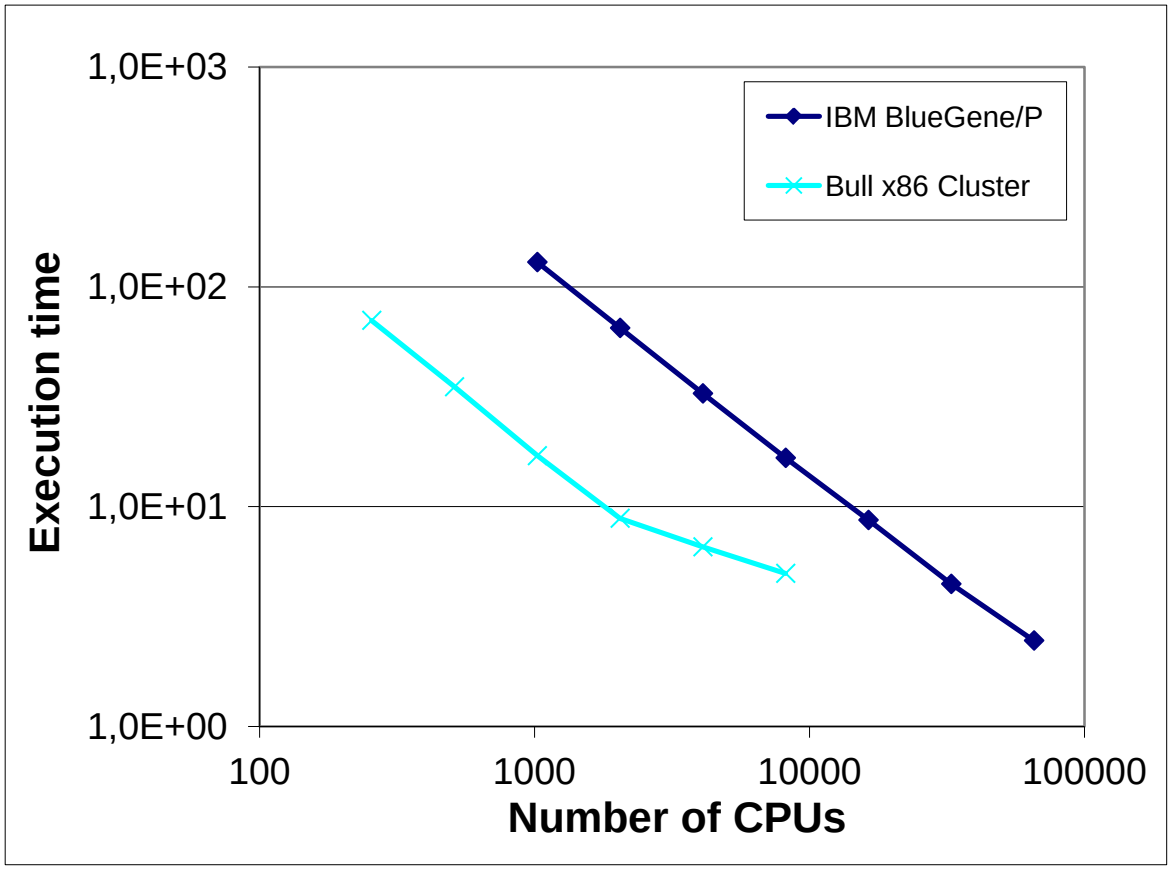

Figure 35: Execution time of QCD, Kernel B

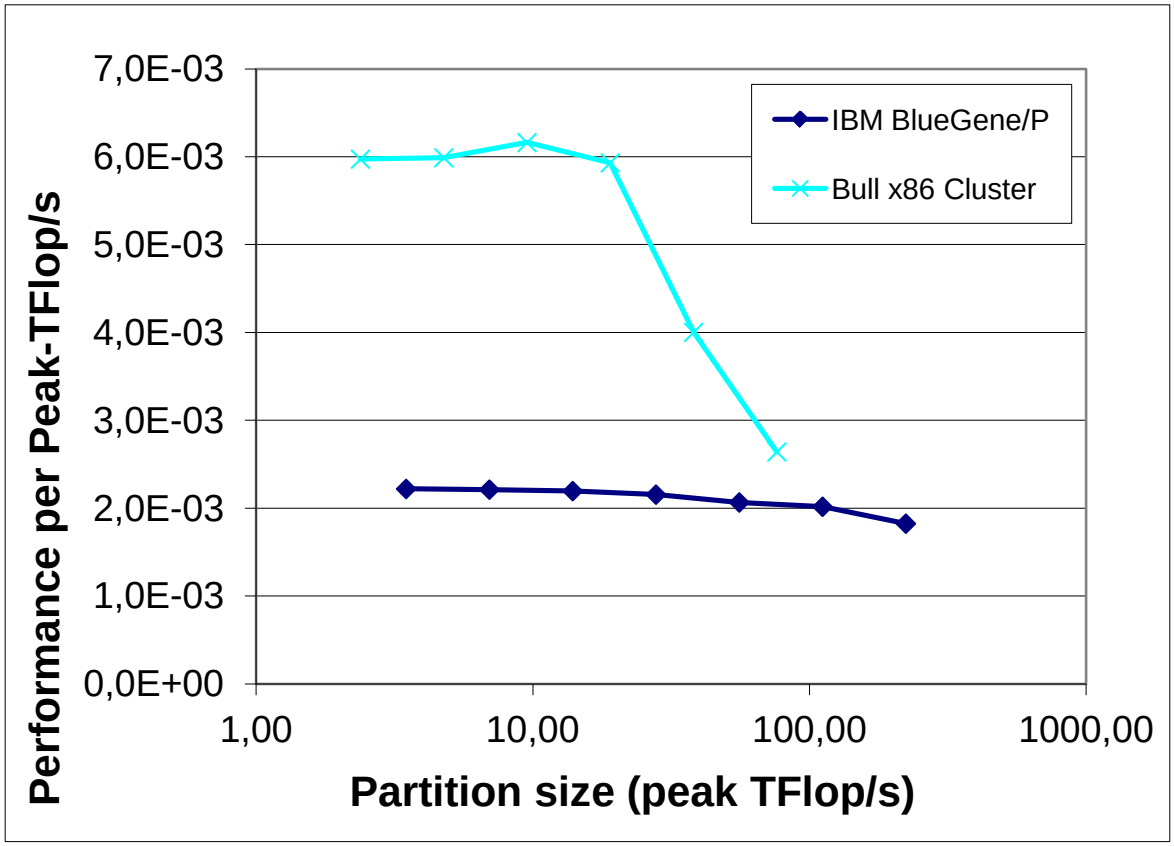

Figure 36: Performance per Peak-TFlop/s for QCD, Kernel B

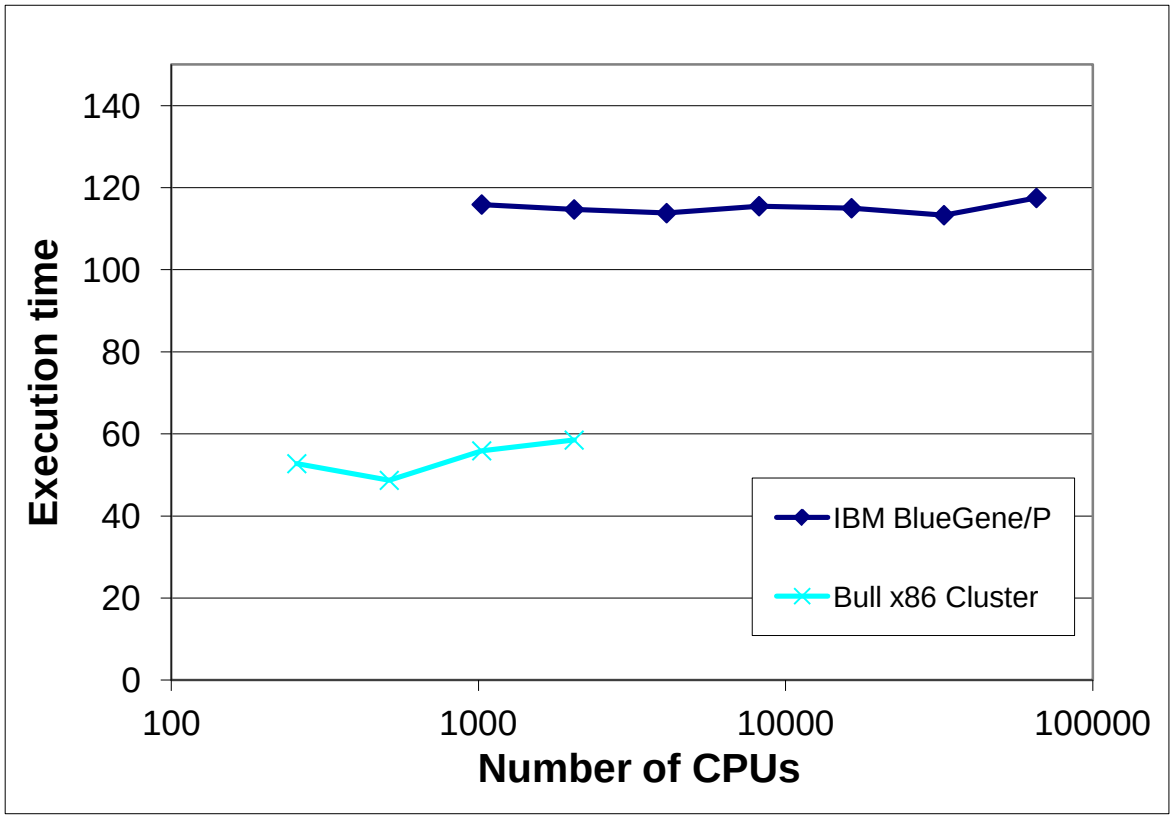

Figure 37: Execution time of QCD, Kernel C

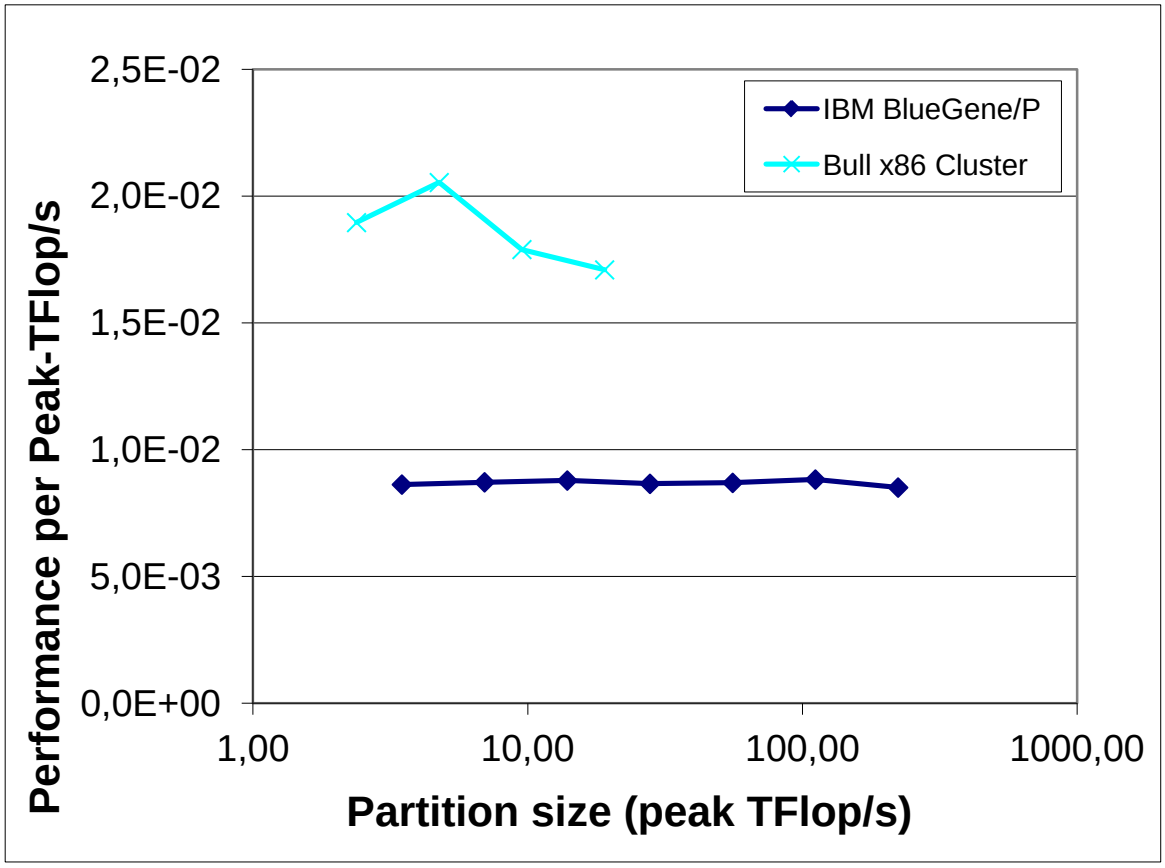

Figure 38: Performance per Peak-TFlop/s for QCD, Kernel C

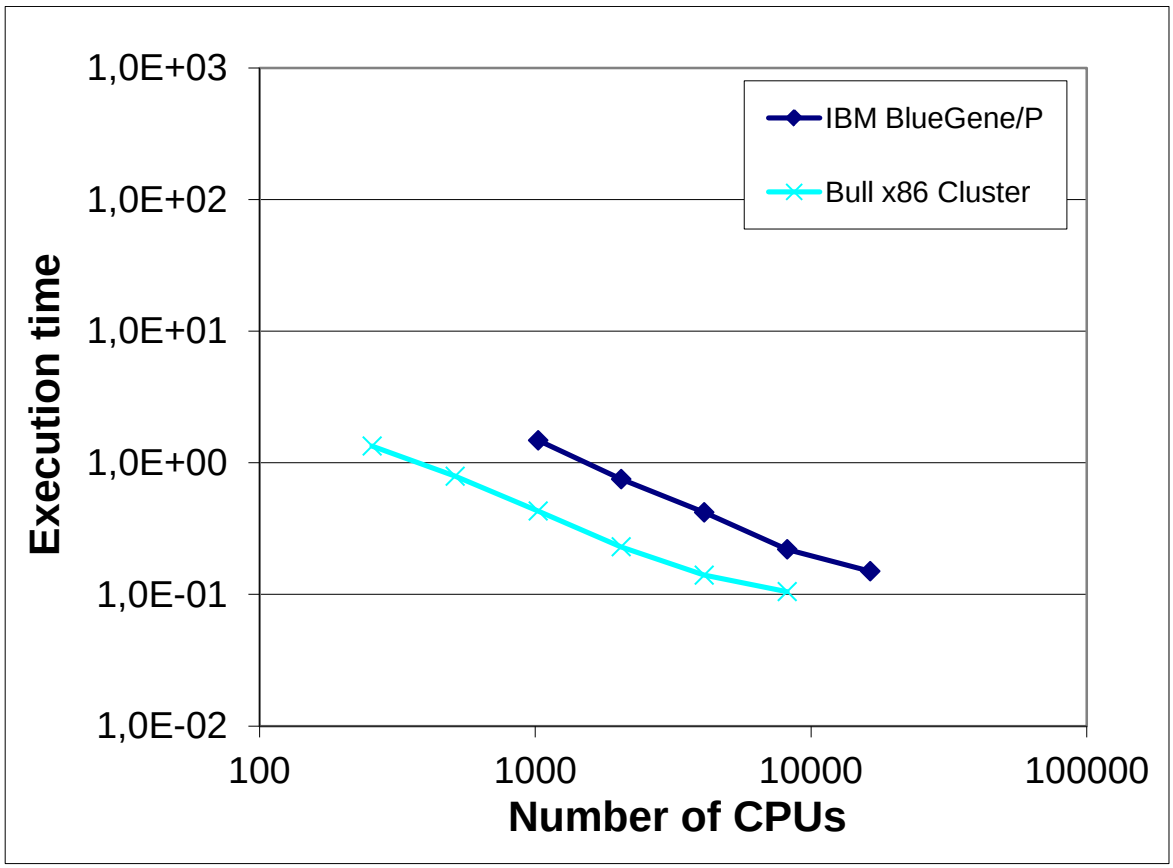

Figure 39: Execution time of QCD, Kernel D

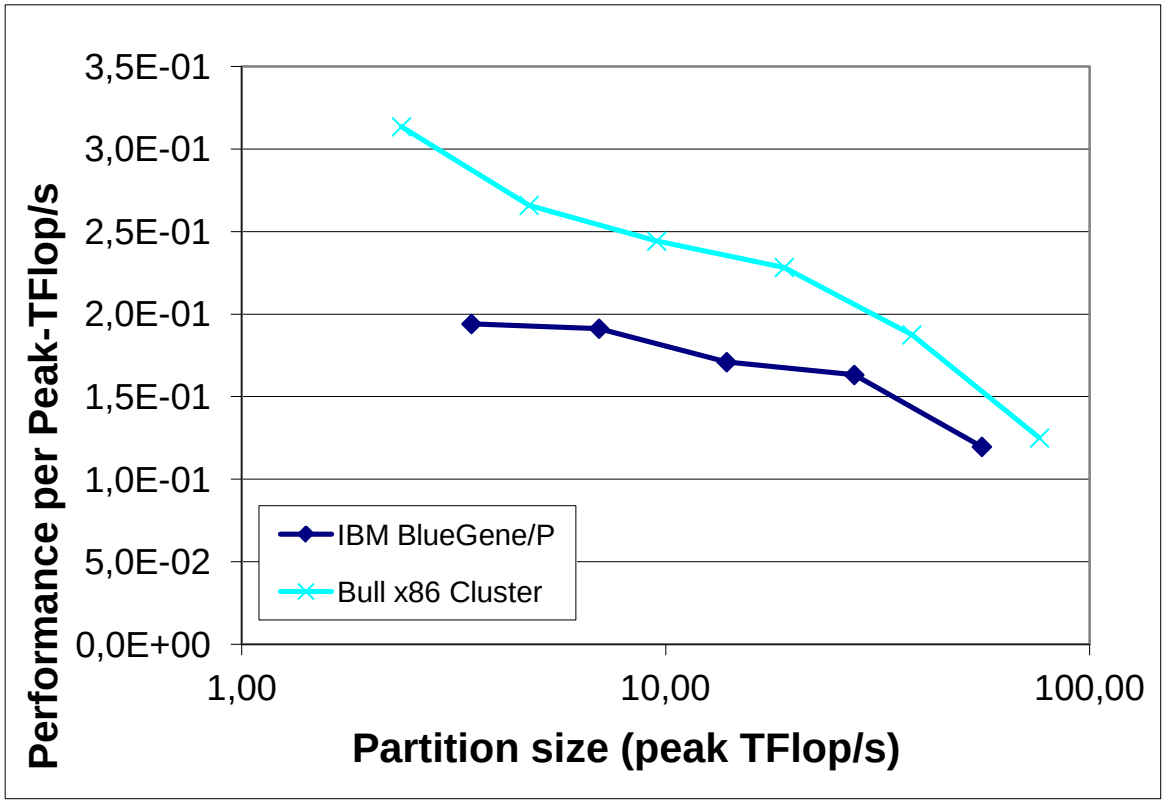

Figure 40: Performance per Peak-TFlop/s for QCD, Kernel D

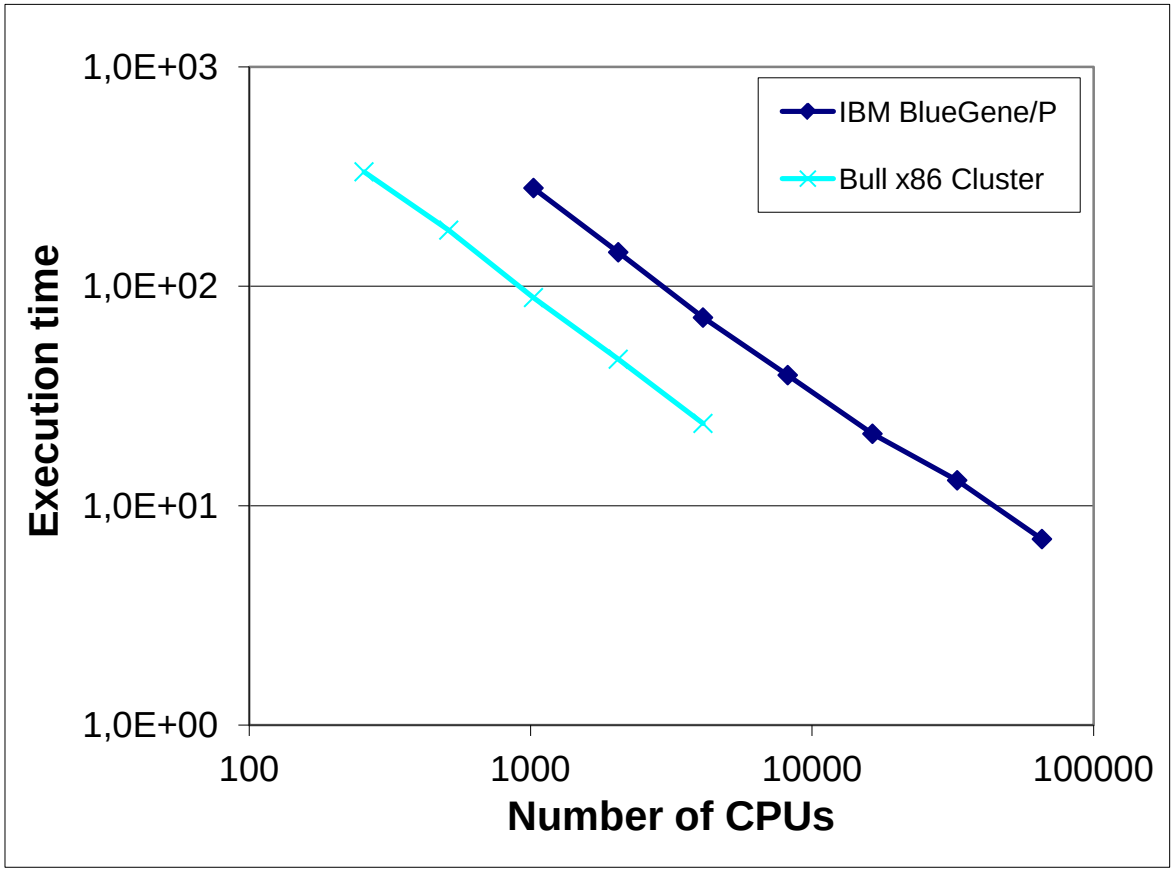

Figure 41: Execution time of QCD, Kernel E

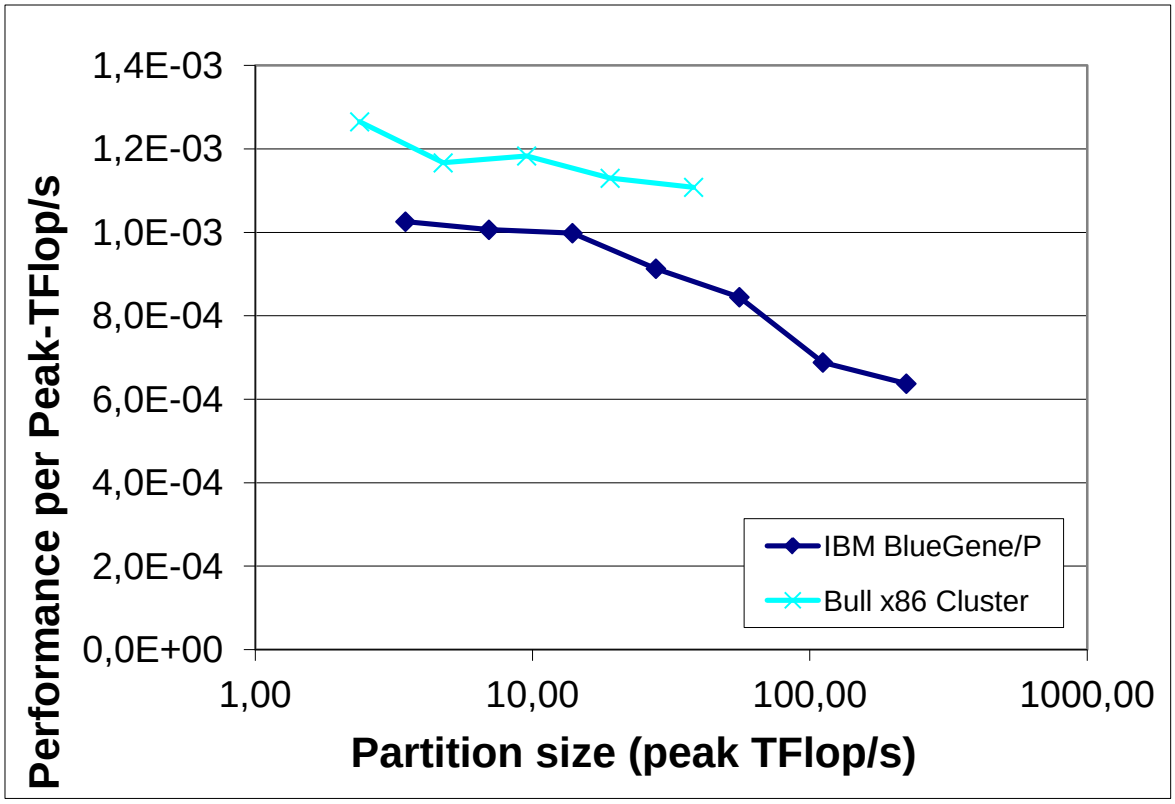

Figure 42: Performance per Peak-TFlop/s for QCD, Kernel E

3.10.4 *Analysis*

The five kernels scale well on both systems under consideration. Because of the faster CPUs on the Bull machine the runs are faster on that machine, but the scaling behaviour is comparable. Some remarks can be made about single kernels: Kernel A is sensitive to the shape used on the IBM BlueGene/P, so the benchmark had to be adapted to the right shape which should be a dense one. Measurements of kernel C were more difficult on the Bull Cluster because the influence of load on the network was bigger than on the IBM BlueGene/P, so runs to the full size of the other kernels could not be shown for that kernel.

3.11 QUANTUM_ESPRESSO

3.11.1 *Summary*

QUANTUM ESPRESSO is an integrated suite of computer codes for electronic-structure calculations and materials modelling at the nanoscale, based on density-functional theory, plane waves, and pseudopotentials (norm conserving, ultrasoft, and Projector Augmented-Wave (PAW)).

3.11.2 *Test Cases*

Test Case A is a free energy simulation of Cu ions in water.

3.11.3 *Results*

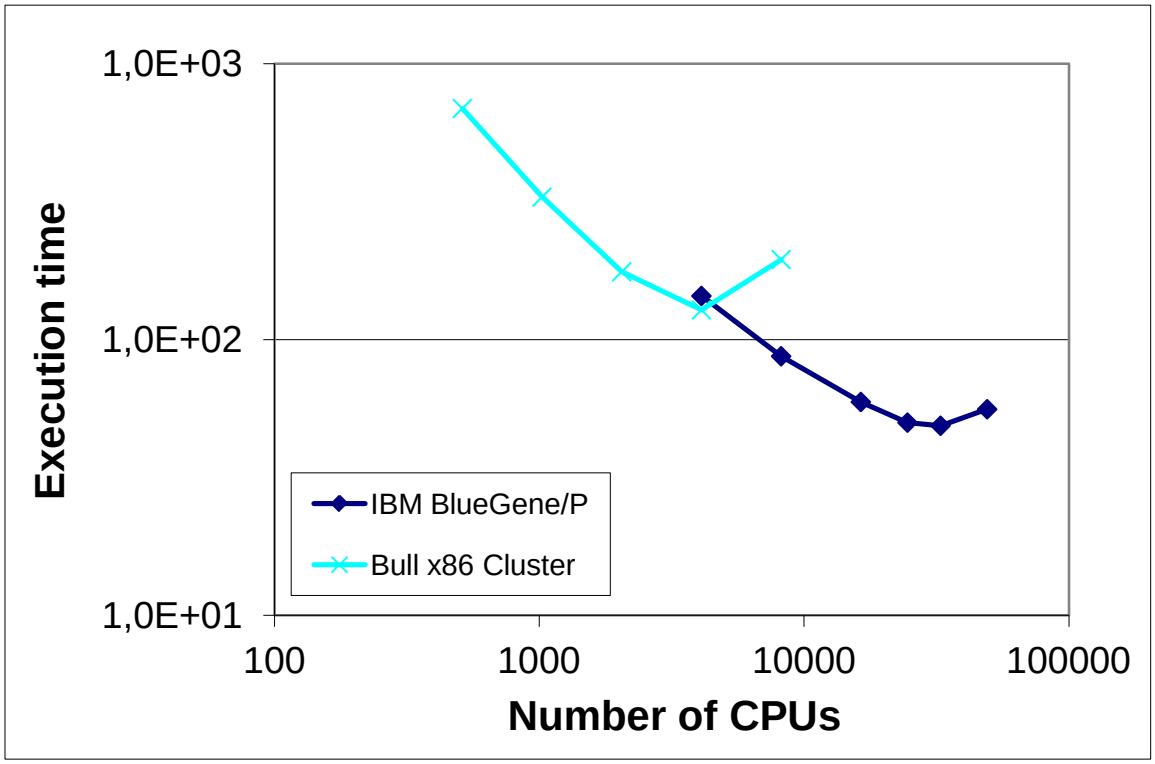

Figure 43: Execution time of Quantum_Espresso, Test Case A

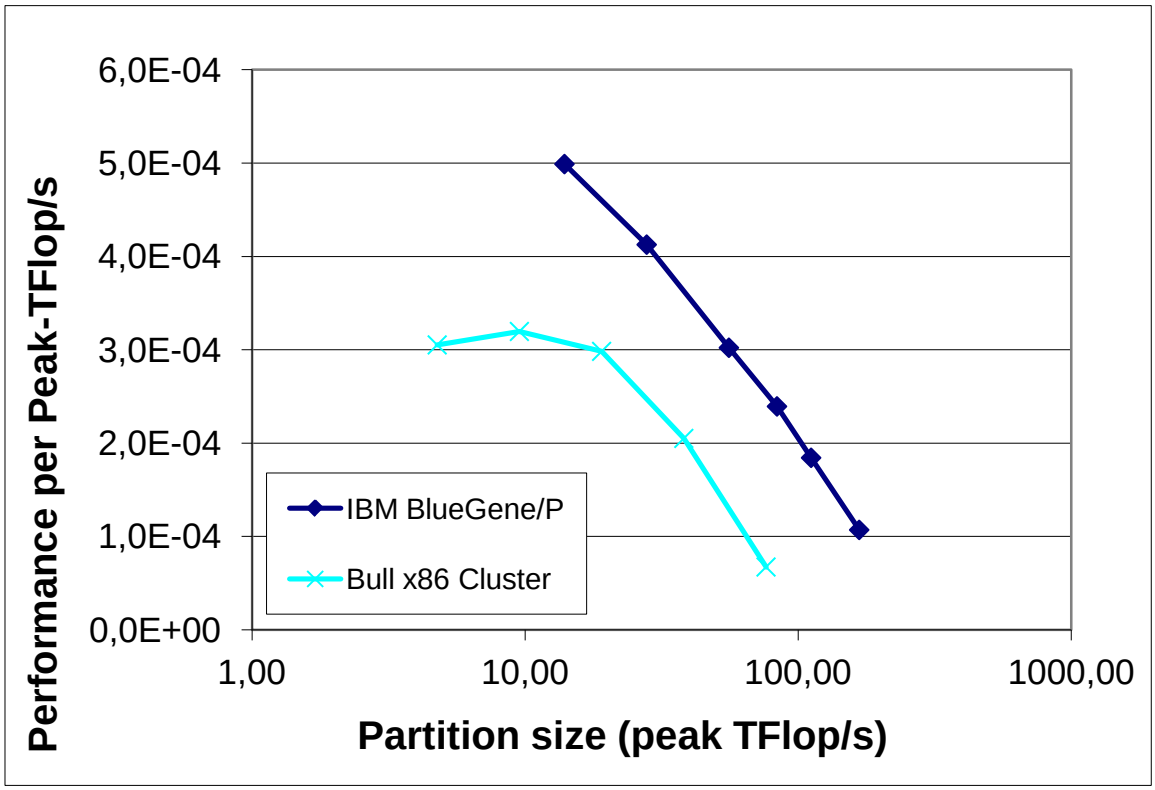

Figure 44: Performance per Peak-TFlop/s for QUANTUM_ESPRESSO, Test Case A

3.11.4 *Analysis*

The simulation of Test Case A uses multi level parallelism to perform the ensemble simulation required to compute free energy. In theory, based on the size of the system and on the two different levels of communication involved, fine grained within each sample and coarse grained between different samples, we would expect a higher scalability with respect to the one observed. A first analysis shows that the saturation observed comes mainly from poor load balancing in the coarse grain communication level rather than saturation in the fine grain communications. This result is very valuable for the code development since it gives some hints regarding future improvements.

3.12 WRF

3.12.1 *Summary*

The Weather Research & Forecasting (WRF) model was developed at the National Center for Atmospheric Research (NCAR) in the United States as a regional- to global-scale model for both research applications and operational weather-forecast systems.

3.12.2 *Test Cases*

Test Case A is a single domain configuration over the North Atlantic and UK consisting of approximately 1200 x 1200 grid points.

3.12.3 *Results*

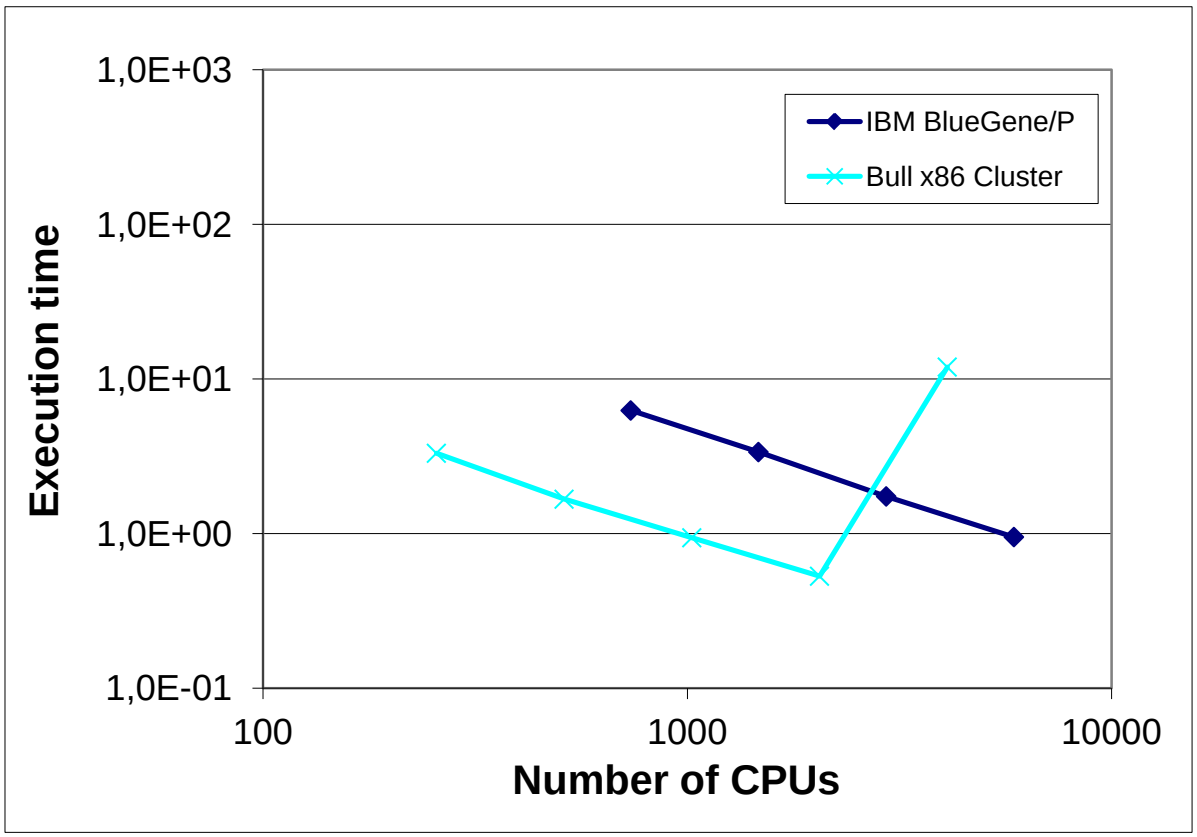

Figure 45: Execution time of WRF, Test Case A

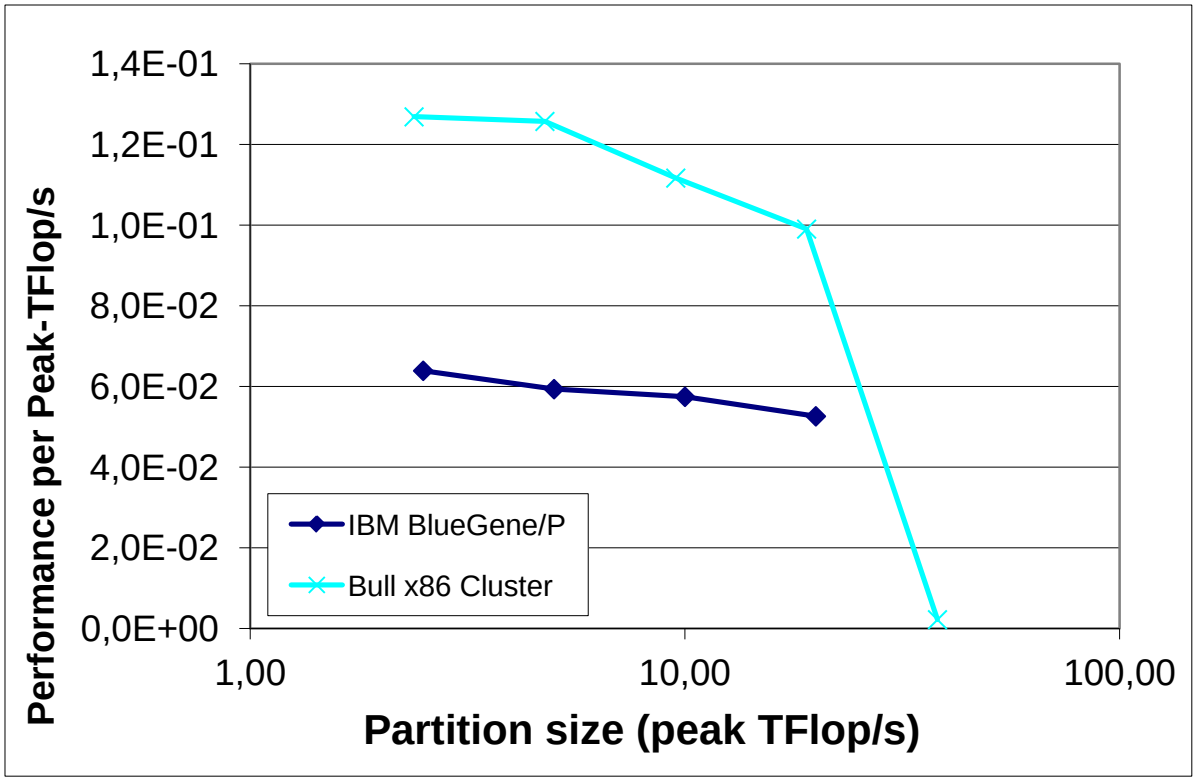

Figure 46: Performance per Peak-TFlop/s for WRF, Test Case A

3.12.4 *Analysis*

The configuration scales well on both the IBM BlueGene/P and the Bull Cluster. However, there appears to be an anomalous slow-down on the Bull Cluster at the largest core count which may be related to a problem with the interconnect on that machine.

3.13 ALYA

3.13.1 *Summary*

ALYA is a Computational Mechanics (CM) code capable of solving different physics, each one with its own modelling characteristics, in a coupled way. Among the problems it solves are: Convection-Diffusion-Reaction, Incompressible Flows, Compressible Flows, Turbulence, Bi-Phasic Flows and free surface, Excitable Media, Acoustics, Thermal Flow, Quantum Mechanics (TDFT) and Solid Mechanics (Large strain).

3.13.2 *Test Cases*

Test Case A is the Nastin test: a mesh with 300^3 linear hexahedral elements.

Test Case B is the Temper test: a mesh with 256^3 linear hexahedral elements.

3.13.3 *Results*

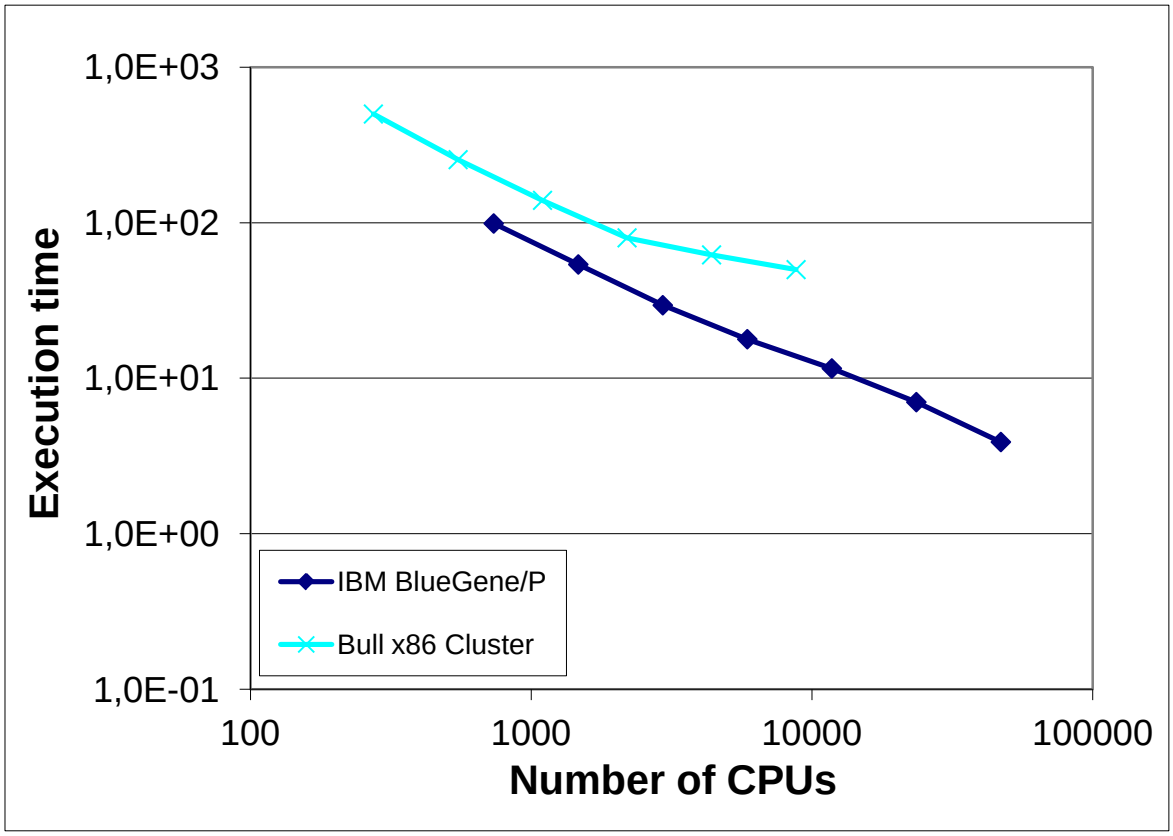

Figure 47: Execution time of ALYA, Test Case A

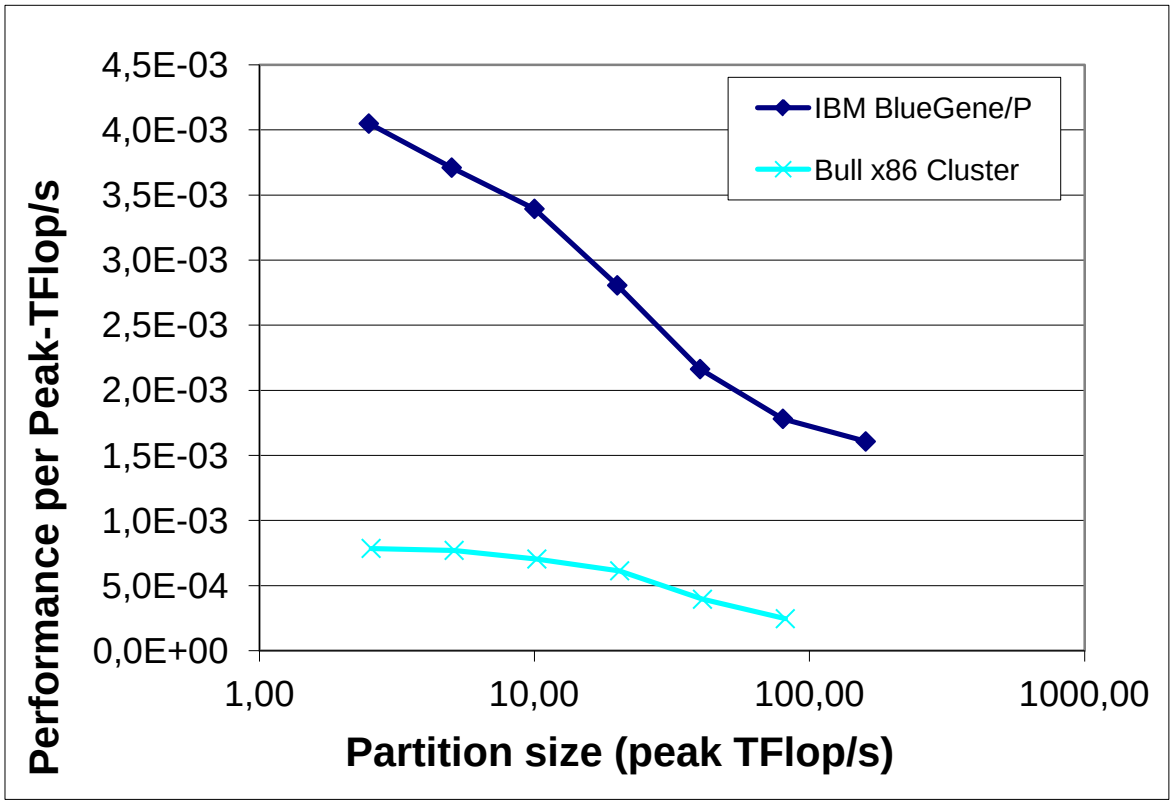

Figure 48: Performance per Peak-TFlop/s for ALYA, Test Case A

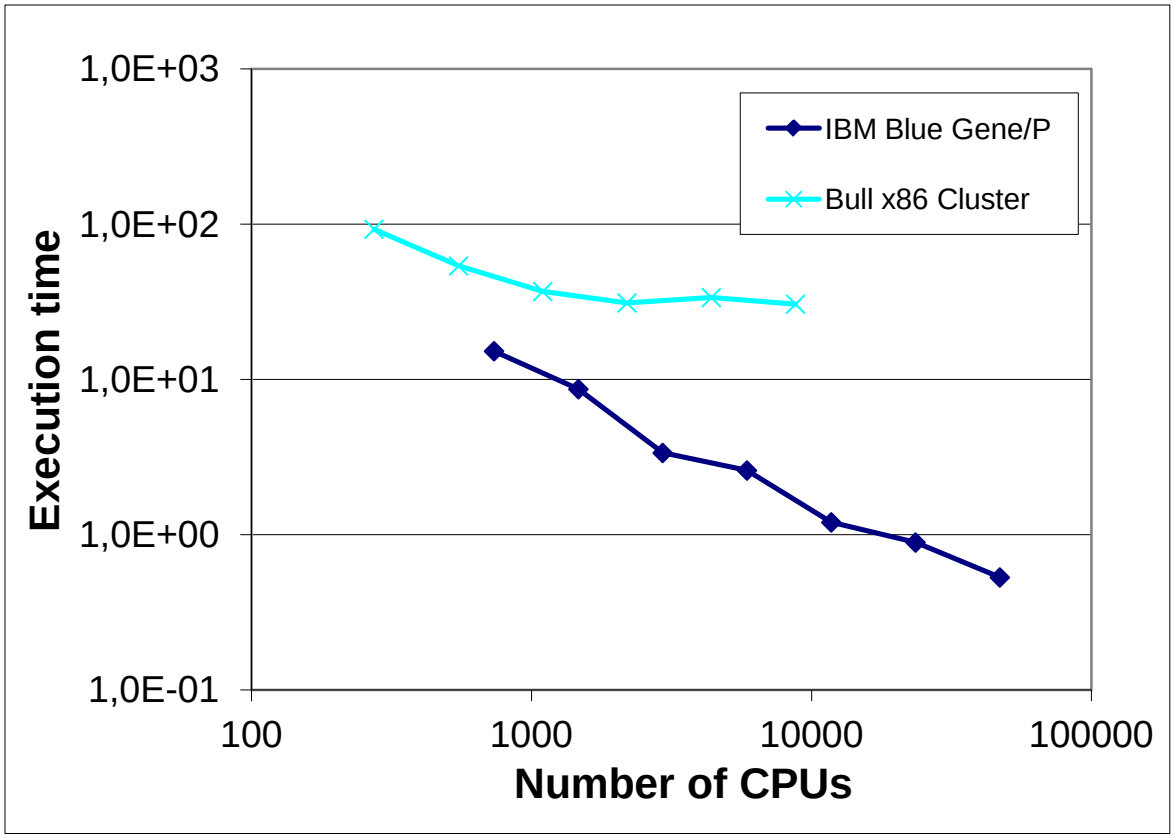

Figure 49: Execution time of ALYA, Test Case B

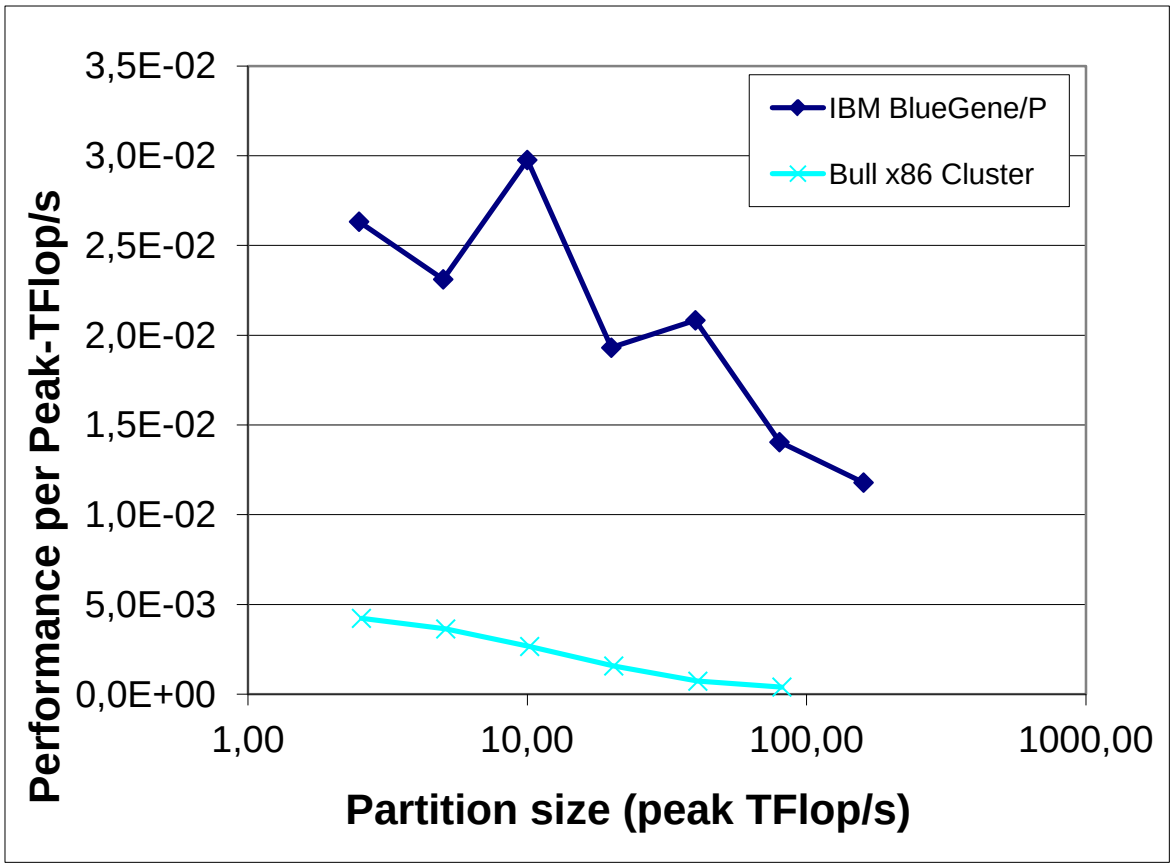

Figure 50: Performance per Peak-TFlop/s for ALYA, Test Case B

3.13.4 *Analysis*

As the results show, Alya scales reasonably well when large problems are simulated. This behaviour can be observed comparing Test Case A (mesh of 300^3 node points) and Test Case B (mesh of 256^3 node points). For Test Case B on JUGENE it shows two times superlinear scaling (peaks in figure 50), this behaviour has been described also in the PRACE whitepaper on ALYA [4].

Future work will focus on running very large cases with hundreds of thousands domains, where serial calls to Metis for domain decomposition have revealed several issues.

3.14 AVBP

3.14.1 *Summary*

AVBP simulates turbulent combustion taking place in turbulent flows within complex geometries. It has been jointly developed in France by CERFACS and IFP to perform Large Eddy Simulation (LES) of reacting flows, in gas turbines, piston engines or industrial furnaces. This compressible LES solver on unstructured and hybrid grids is employed in multiple configurations for industrial gas turbines, aero gas turbines, rocket engines, and laboratory burners used to study unsteady combustion. AVBP is a massively parallel CFD ode that solves laminar and turbulent compressible reacting flows.

3.14.2 *Test Cases*

Test Case A is a simulation containing 37 million cells.

3.14.3 *Results*

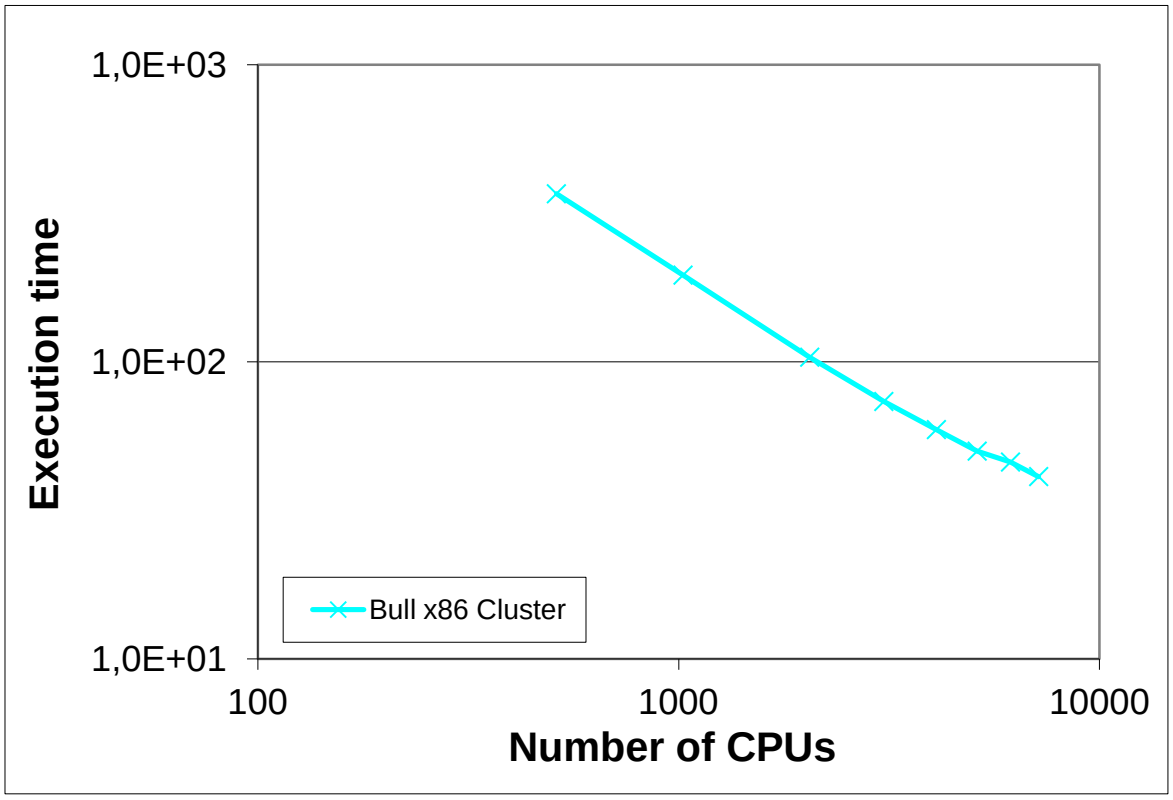

Figure 51: Execution time of AVBP, Test Case A

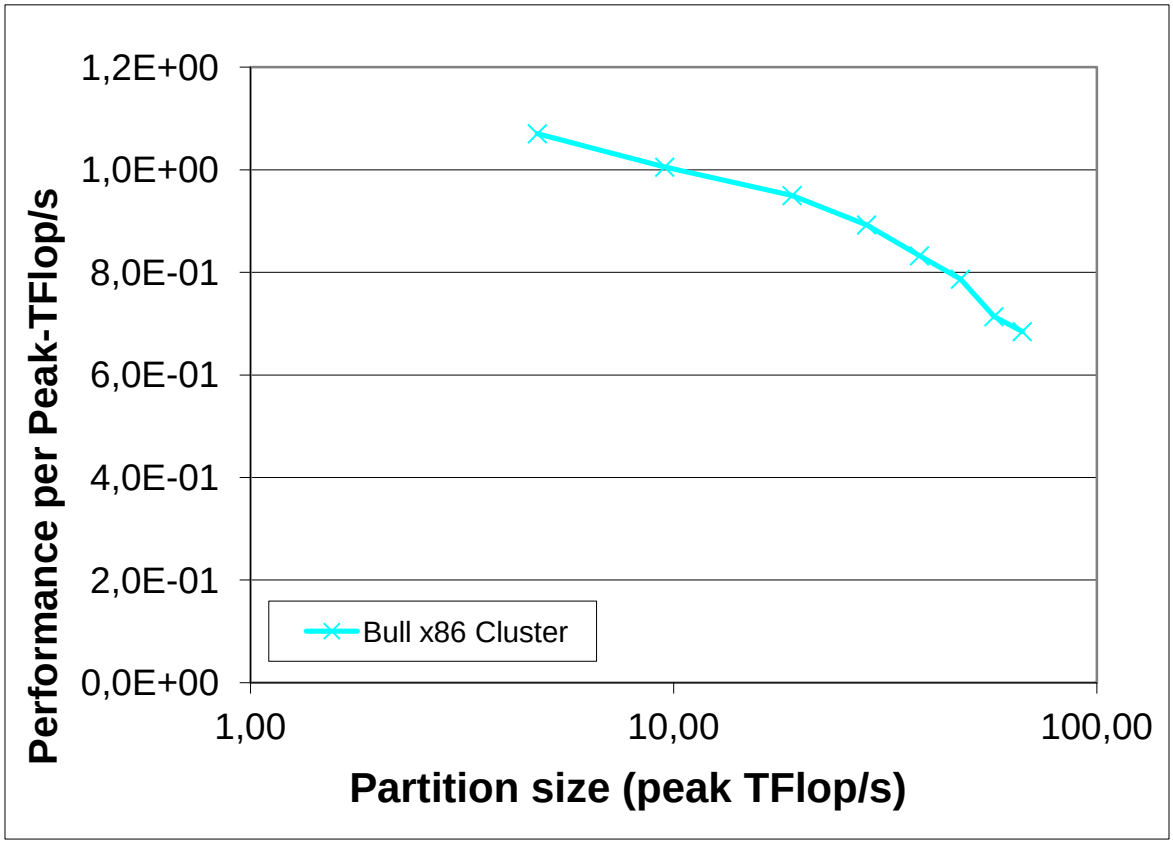

Figure 52: Performance per Peak-TFlop/s for AVBP, Test Case A

3.14.4 *Analysis*

This test case scales very well on the Bull x86 Cluster up to 67 TFlop/s.

3.15 ELMER

3.15.1 *Summary*

Elmer is a finite element software package for the solution of partial differential equations. Elmer can deal with a great number of different equations, which may be coupled in a generic manner making Elmer a versatile tool for multiphysical simulations.

3.15.2 *Test Cases*

Test Case A is a finite element model for heat conduction in a three dimensional solid.

3.15.3 *Results*

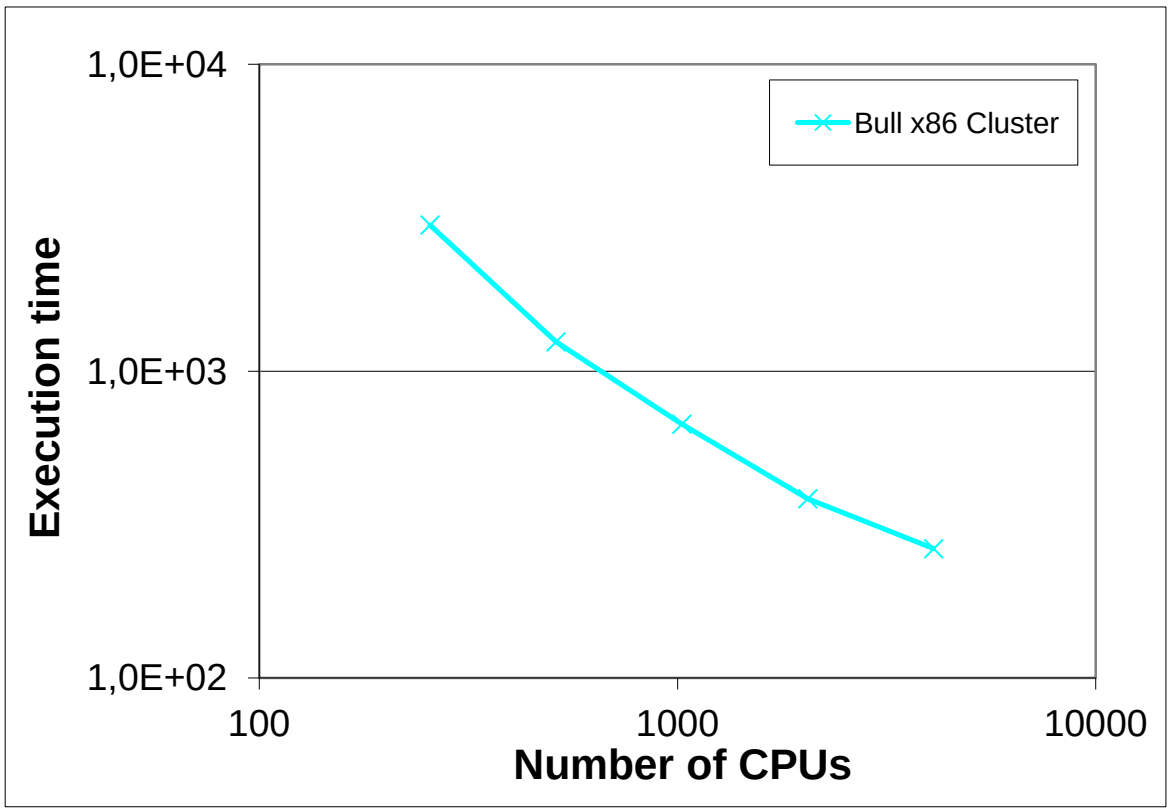

Figure 53: Execution time of ELMER, Test Case A

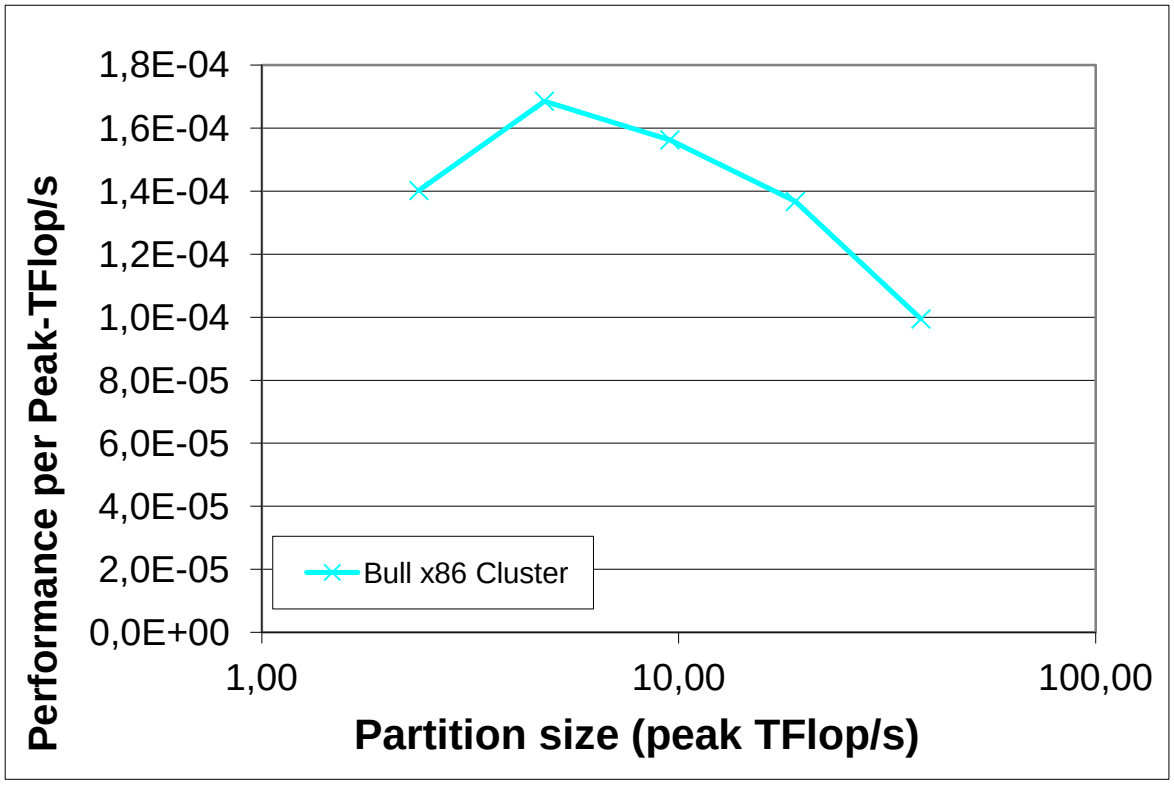

Figure 54: Performance per Peak-TFlop/s for ELMER, Test Case A

3.15.4 *Analysis*

On CURIE the scalability observed within the test runs is compatible with the grid size, data partition strategy implemented (four mesh levels) and linear iterative solver method used (Biconjugate Gradient stabilized method). Higher scalability can be obtained by changing the dataset and/or linear solver details such as turning off the pre-conditioner. Unfortunately it was not possible to execute the test on JUGENE, since we found a compatibility issue with the dynamic loader used by the ELMER solver.

3.16 GPAW

3.16.1 *Summary*

GPAW is a software package for electronic structure simulations within the density-functional theory (DFT). DFT allows studies of ground state properties of nanoscale systems in materials science and quantum chemistry. In addition to basic DFT, GPAW implements also the time-dependent density-functional theory (TDDFT) both in real-time propagation and linear response formalisms. With TDDFT, also excited state properties can be studied.

GPAW can use several parallelizetion levels. The basic parallelization strategy is domain decomposition of the real-space grids. Domain decomposition involves only nearest neighbour communication with modest amount of data (typical message sizes are few tens of kB). In periodic systems, nearly trivial and very scalable parallelization over so called k-points is possible, and in magnetic systems very similar parallelization over spin can be performed. Additionally, parallelization over electronic states is possible.

3.16.2 *Test Cases*

Test Case A is a medium sized system consisting of 256 water molecules, there are thus 768 atoms and 2048 valence electrons (1056 electronic states are explicitly treated). A ground state DFT calculation (with only a few iterations) is performed. Parallelization is over real-space domains, electronic state parallelization is also included when the number of CPU cores is larger than 512. This test case is included mainly as comparison to the PRACE Preparatory Project where the same test case was used.

Test Case B is a larger scale system consisting of 702 Si atoms, 1520 electronic states are explicitly treated. A ground state calculation with few iterations is performed. Parallelization is over real-space domains and electronic states.

Test Case C is a linear response TDDFT calculation of the same Si cluster as in case B. Omega matrix is constructed parallelizing both over real-space domains and electron-hole pairs.

Test Case D is a real-time TDDFT calculation of the Si cluster. Few time-propagation steps are performed, parallelization is over real-space domains and electronic states.

3.16.3 *Results*

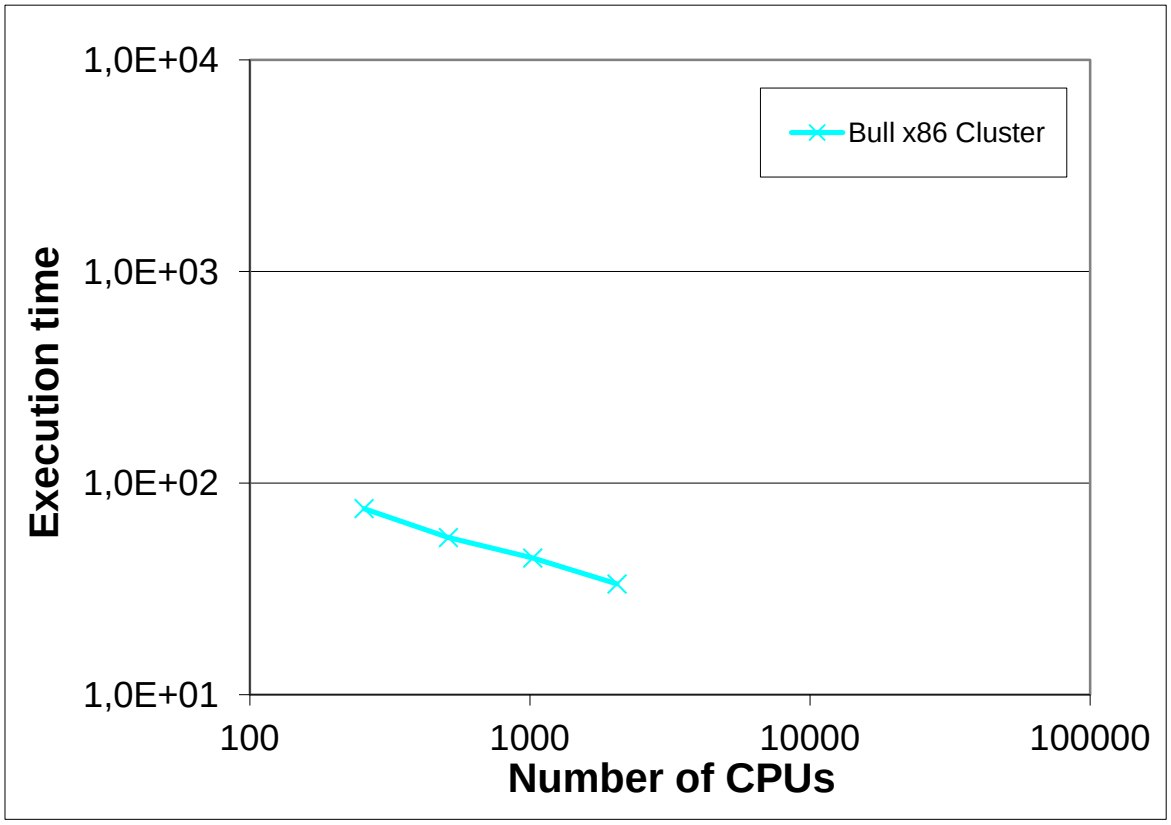

Figure 55: Execution time of GPAW, Test Case A

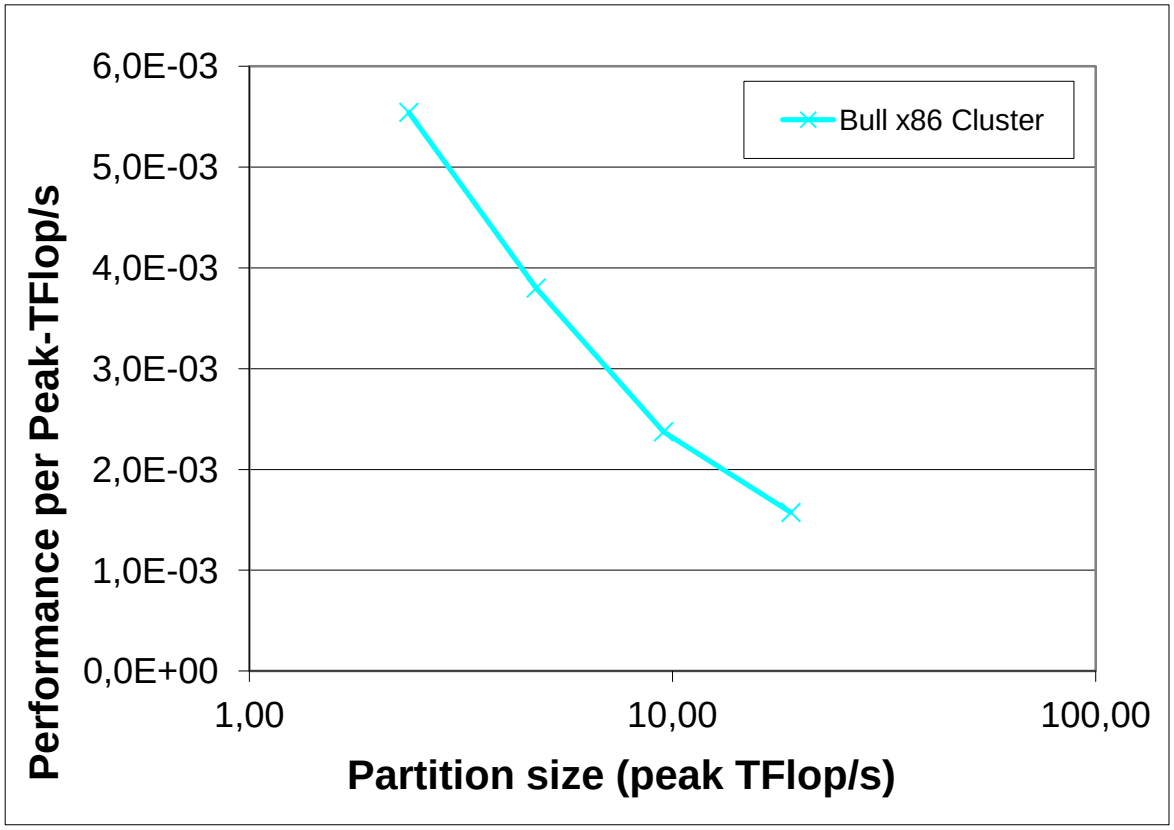

Figure 56: Performance per Peak-TFlop/s for GPAW, Test Case A

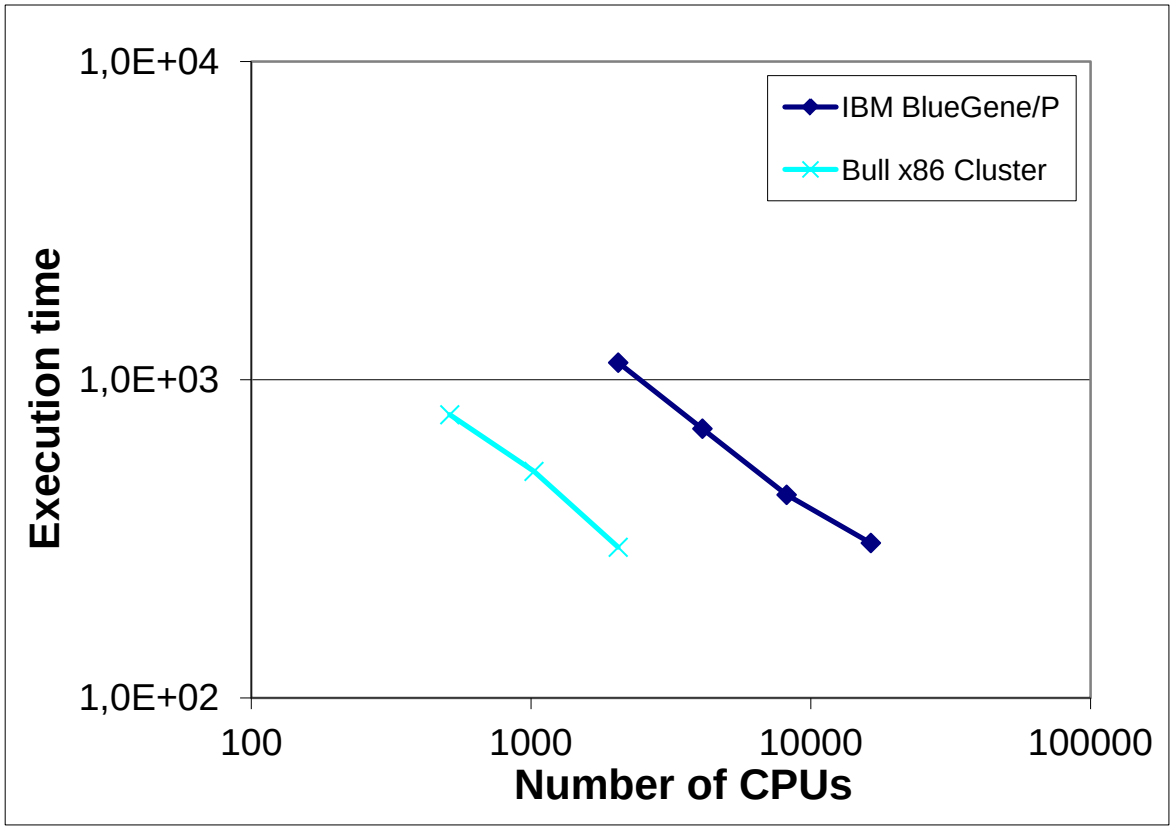

Figure 57: Execution time of GPAW, Test Case B

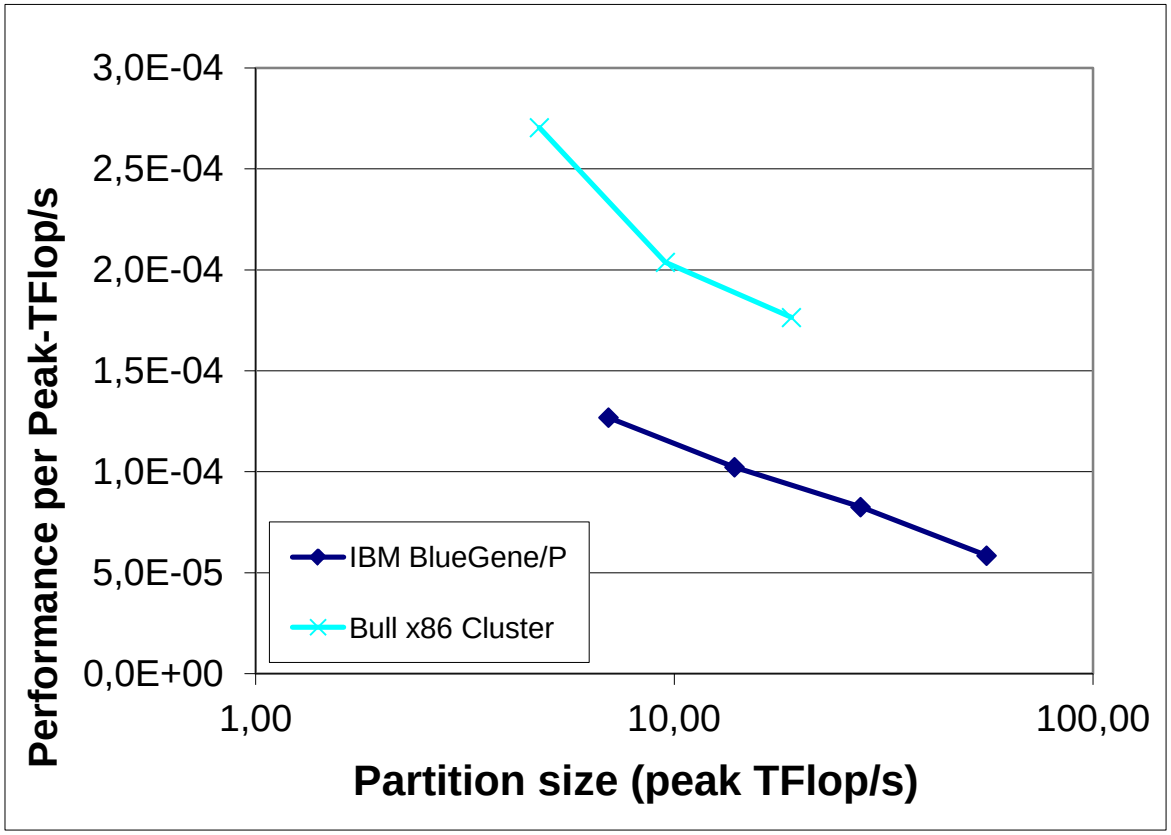

Figure 58: Performance per Peak-TFlop/s for GPAW, Test Case B

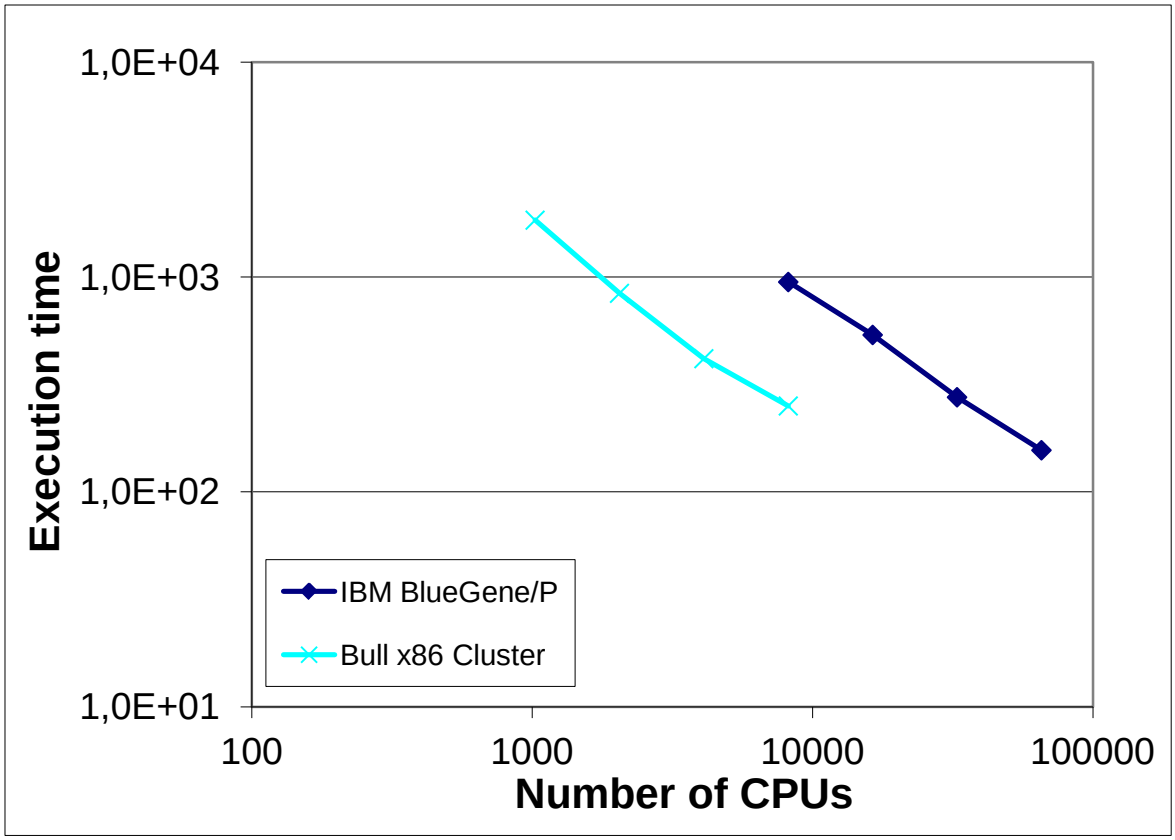

Figure 59: Execution time of GPAW, Test Case C

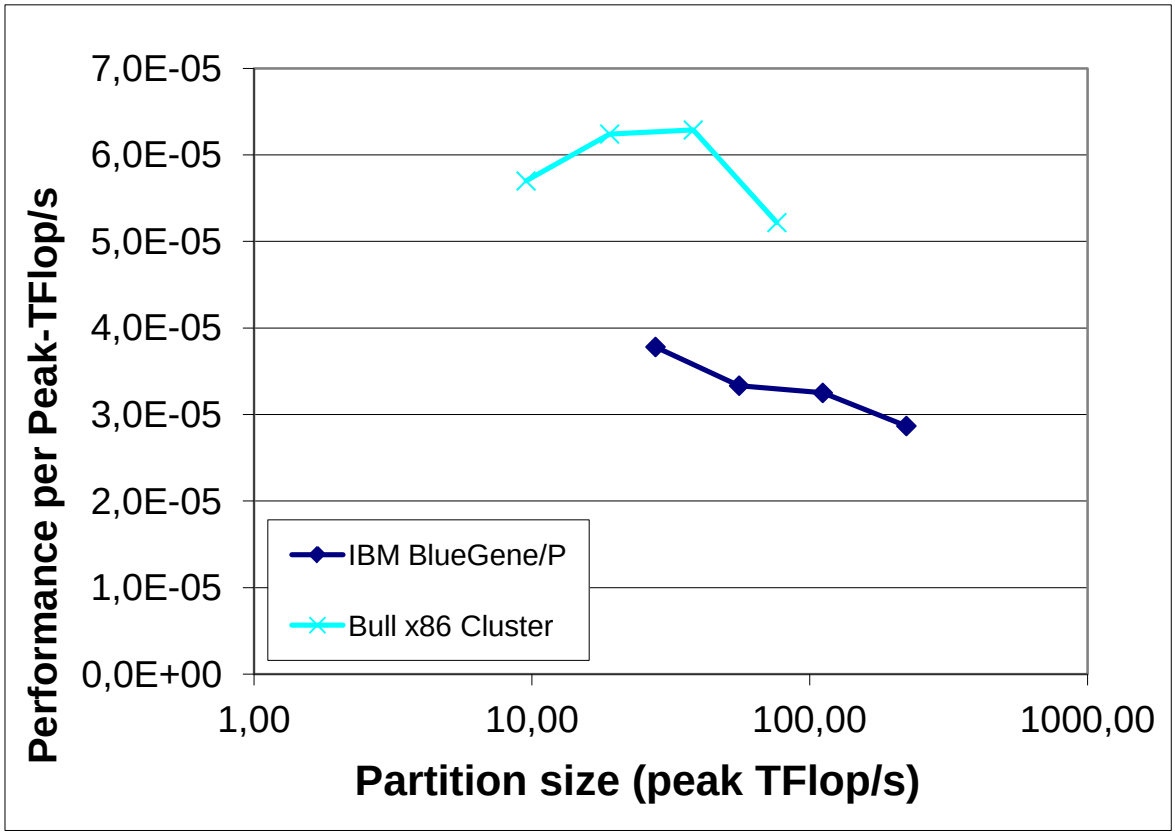

Figure 60: Performance per Peak-TFlop/s for GPAW, Test Case C

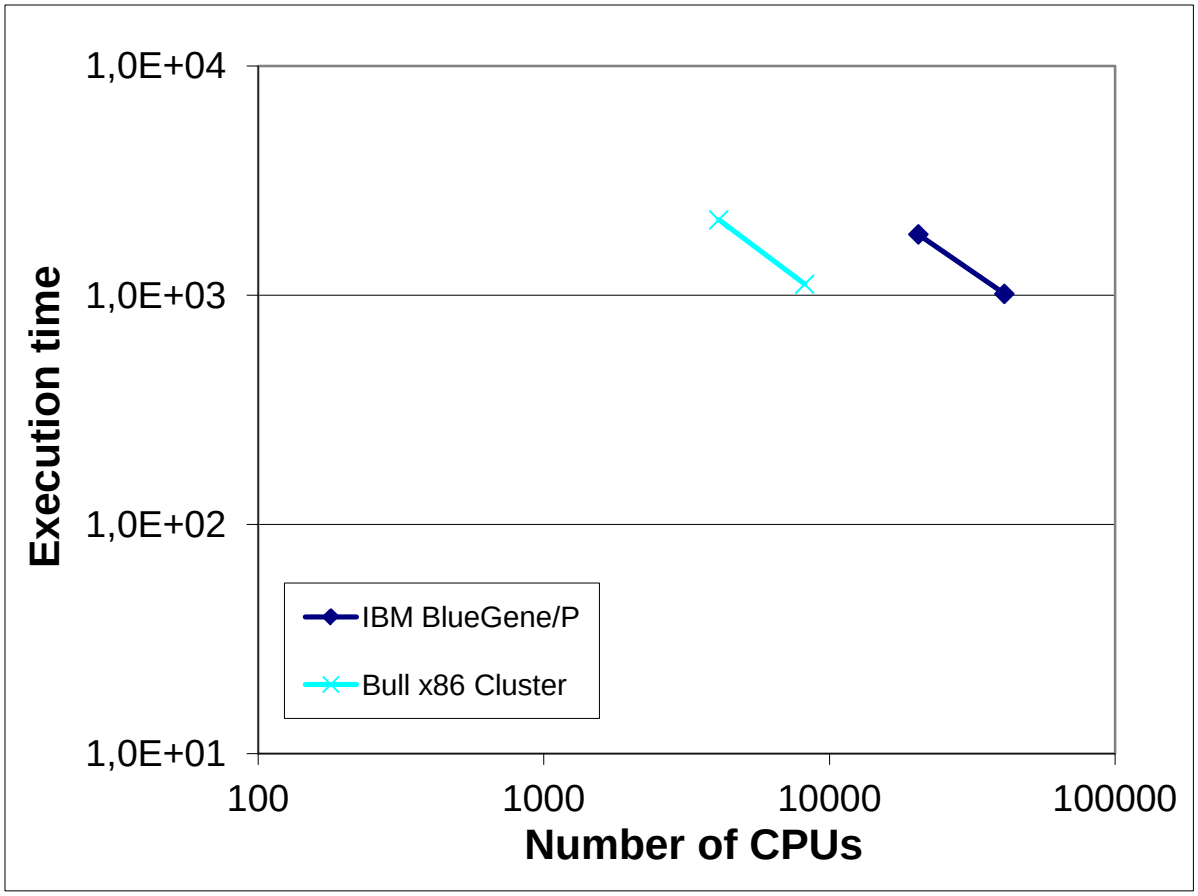

Figure 61: Execution time of GPAW, Test Case D

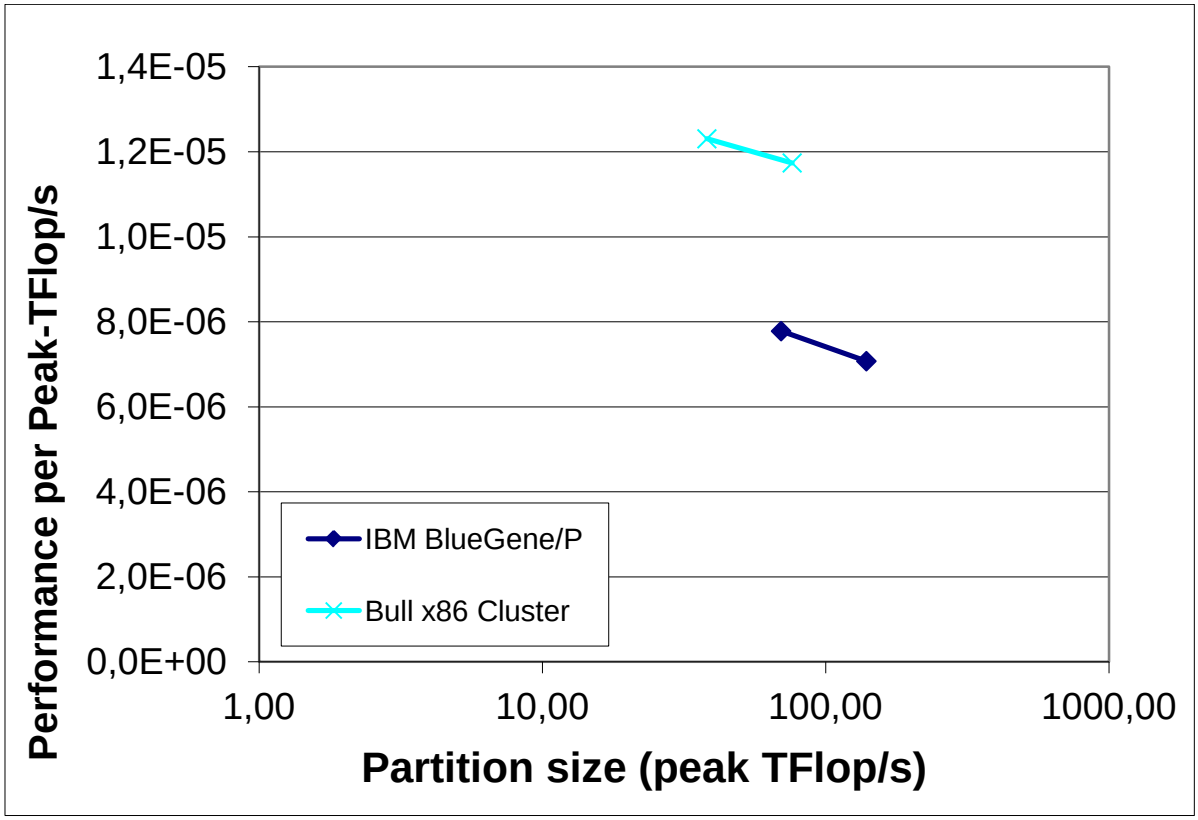

Figure 62: Performance per Peak-TFlop/s for GPAW, Test Case D

3.16.4 *Analysis*

The scaling of Test Case A on CURIE is poor. In the PRACE Preparatory Project benchmarking, the same input data showed a speed-up between 1024 and 2048 CPU cores in the range of 1.55-1.77 in various systems (Louhi Cray XT5, JUGENE, Huygens).

Test case B scales relatively wellup to 8192 CPU cores on JUGENE. On CURIE scaling is good up to 2048 CPU cores which is the maximum number utilized in this test.

The parallelization over electron-hole pairs and electronic states in TDDFT calculations of case C and case D are significantly less communication intensive than the electronic state parallelization in ground state calculations. Consequently, the parallel scaling remains excellent to very high numbers of CPU cores, both on CURIE and JUGENE. It is expected that with a modest increase in the size of the input data set, scaling to the full JUGENE system would be possible.

3.17 HELIUM

3.17.1 *Summary*

HELIUM simulates the behaviour of helium atoms using time-dependent solutions of the full-dimensional Schrödinger equation.

3.17.2 *Test Cases*

Test Case A is a simulation using 3060 x 3060 blocks and L_Max=16. The Test Case A was used on the IBM BlueGene/P.

Test Case B is a simulation using 1540x1540 blocks and L_Max=16. The Test Case B was used on the Bull x86 Cluster.

3.17.3 *Results*

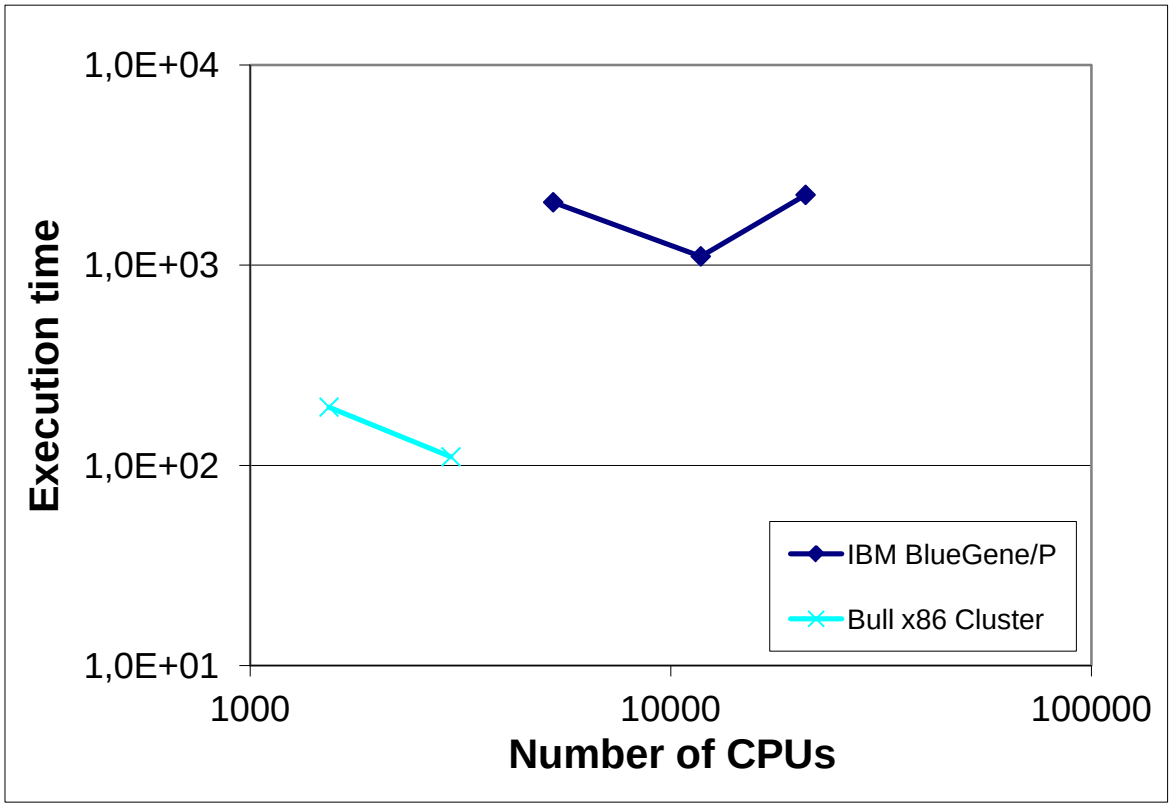

Figure 63: Execution time of Helium, Test Case A for IBM BlueGene/P, Test Case B for Bull x86 Cluster

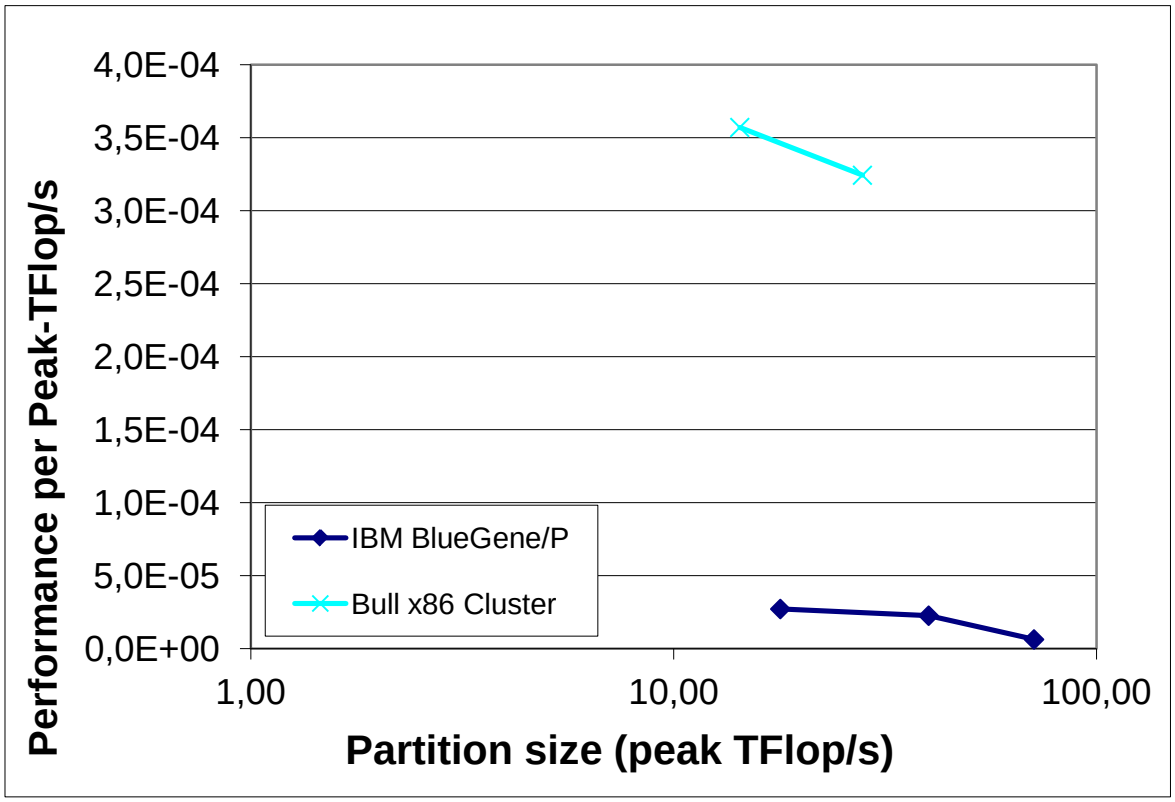

Figure 64: Performance per Peak-TFlop/s for Helium, Test Case A for IBM BlueGene/P, Test Case B for Bull x86 Cluster

3.17.4 *Analysis*

Test Case A can scale up to around 40 TFlop/s peak on IBM BlueGene/P, but then the scaling tails off when increasing to larger partition sizes. Test Case B can scale up to around 40 TFlop/s peak on the Bull x86 Cluster. However, the larger size Test Case A failed to run on the Bull x86 Cluster. Both test cases have relatively low efficiency on the two systems.

3.18 OCTOPUS

3.18.1 *Summary*

Octopus simulates complex electronic processes in medium to large systems, using Density-Functional Theory (DFT) and in particular its time-dependent formulation.

3.18.2 *Test Cases*

Test Case A is a simulation of a chlorophyll fragment with 650 atoms.

3.18.3 *Results*

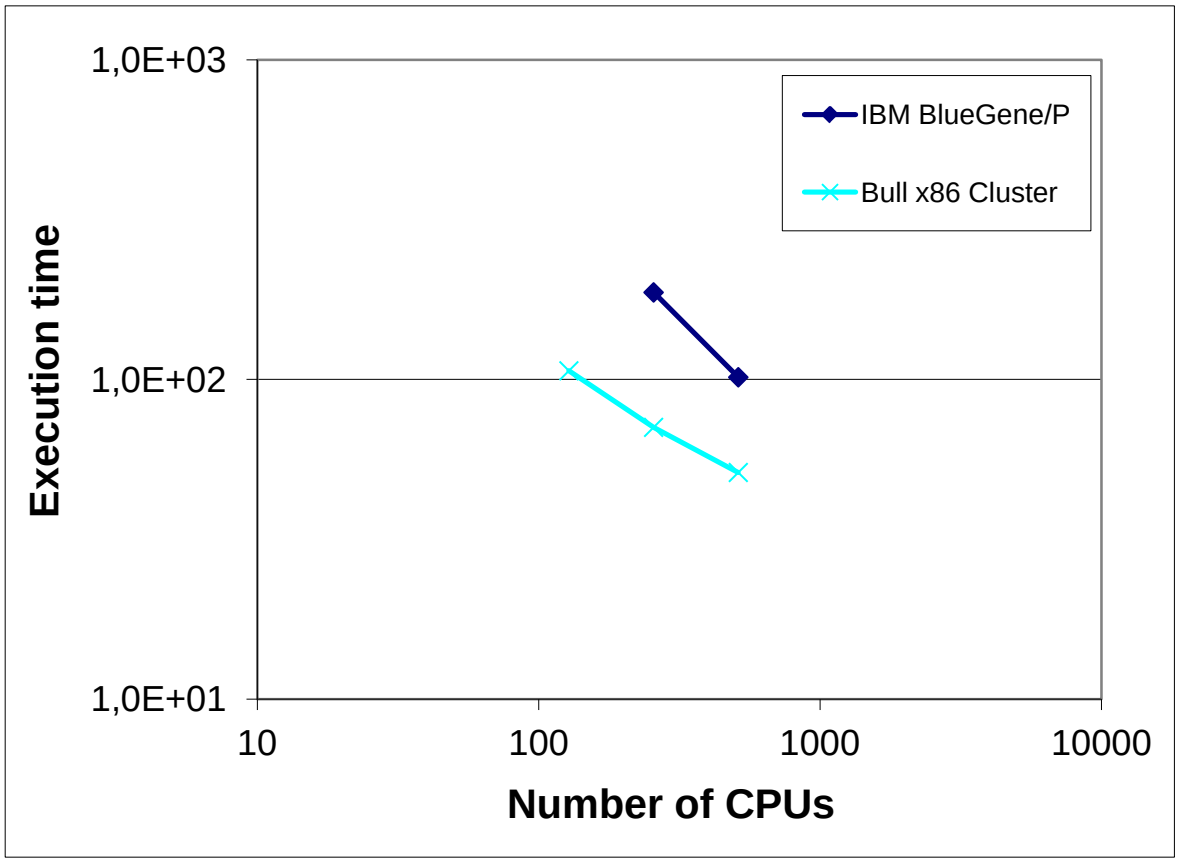

Figure 65: Execution time of OCTOPUS, Test Case A

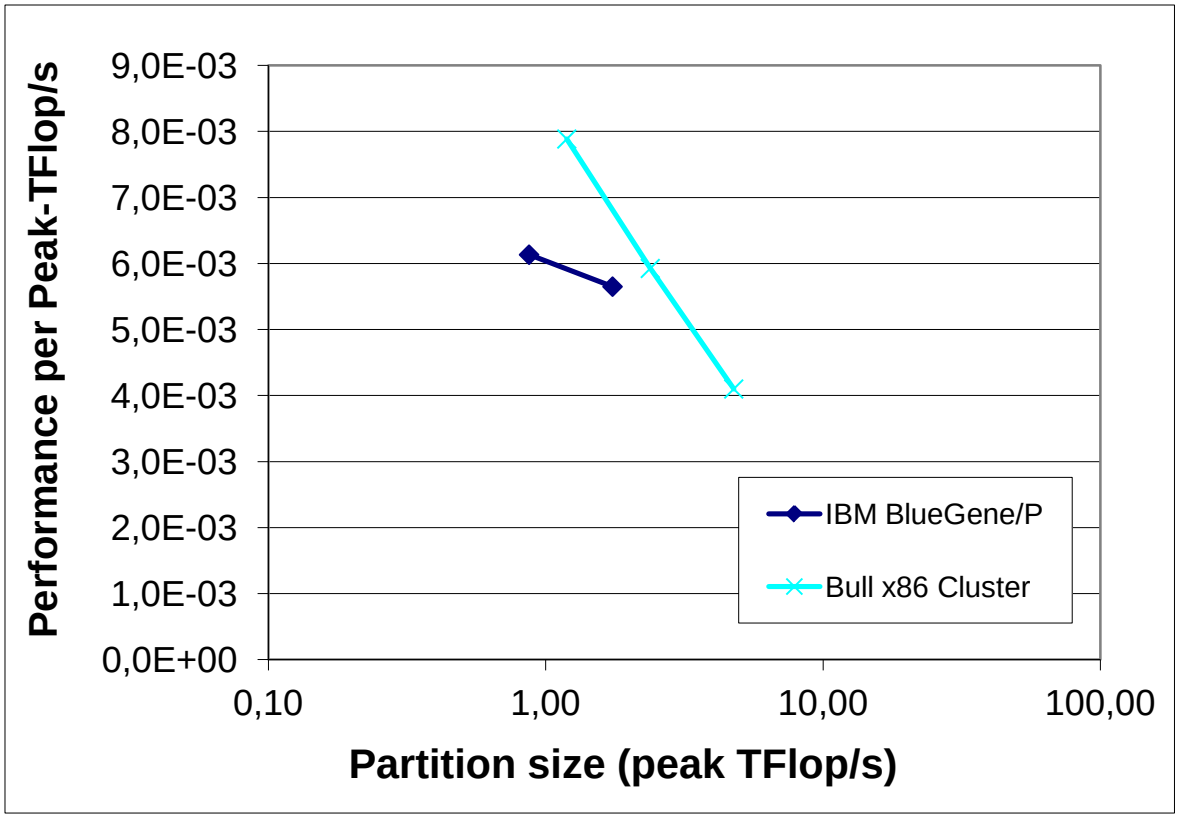

Figure 66: Performance per Peak-TFlop/s for OCTOPUS, Test Case A

3.18.4 *Analysis*

In Octopus, to perform a time-dependent calculation requires a previous run to obtain the ground-state. Unfortunately the later (ground state) has a much worse scaling than the former, so the results presented are for the time-dependent part only.

Furthermore, the poor scaling of the ground-state part limited the size of the partitions used for this benchmark. The runs performed in both systems show good scaling, but runs with a larger number of cores would be desirable.

3.19 PEPC

3.19.1 *Summary*

PEPC is a parallel tree-code for computation of long-range Coulomb forces. The forces are calculated based on the Barnes-Hut algorithm. The code takes advantage of multipole-groupings of distant particles to reduce the original $O(N^2)$ scaling of the calculation to an O(N log N) scaling.

3.19.2 *Test Cases*

Test Case A is a simulation with 8×10^6 particles.

Test Case B is a simulation with 128×10^6 particles.

Test Case C is a simulation with 2048×10^6 particles.

3.19.3 *Results*

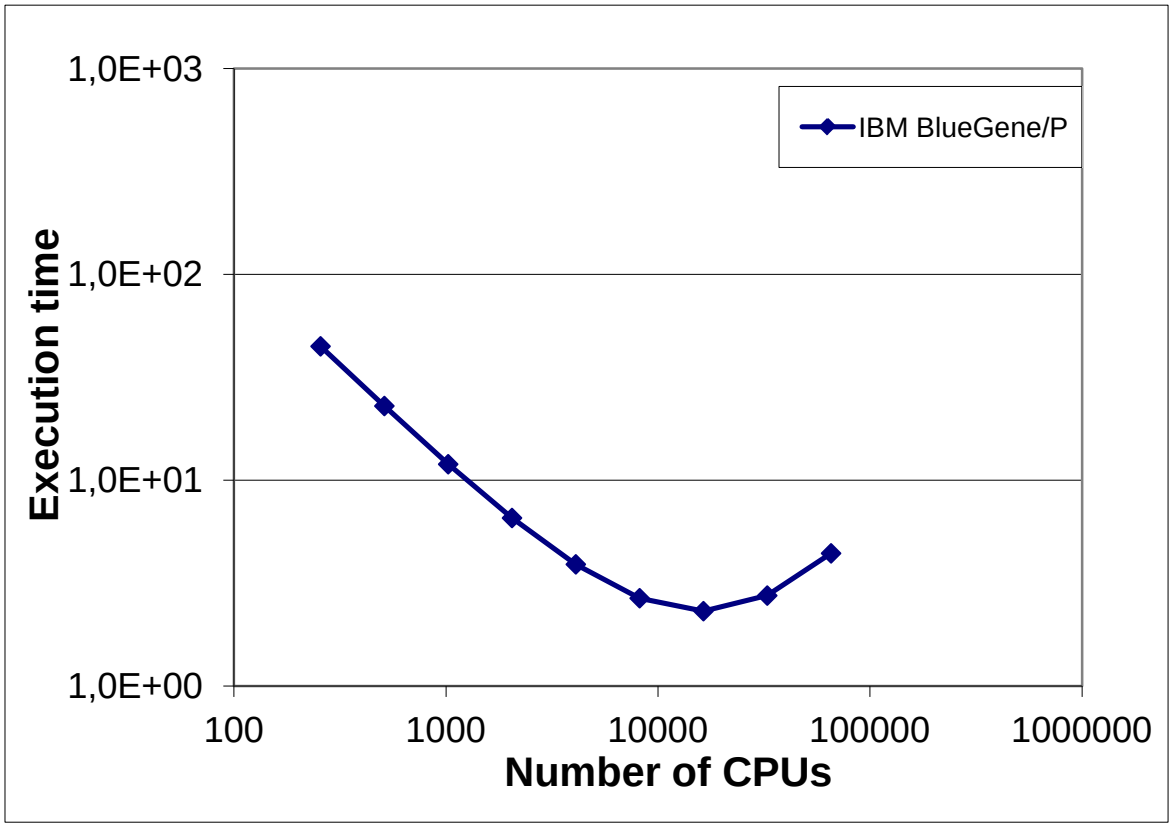

Figure 67: Execution time of PEPC, Test Case A

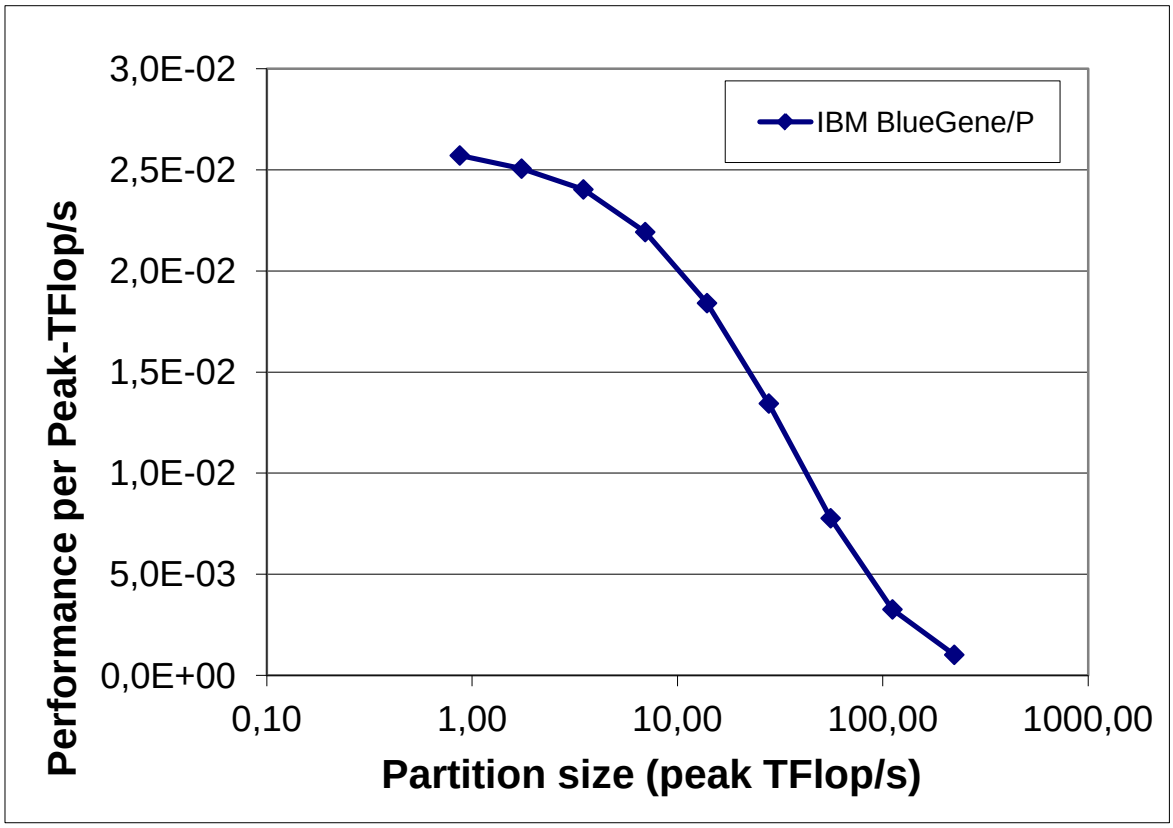

Figure 68: Performance per Peak-TFlop/s for PEPC, Test Case A

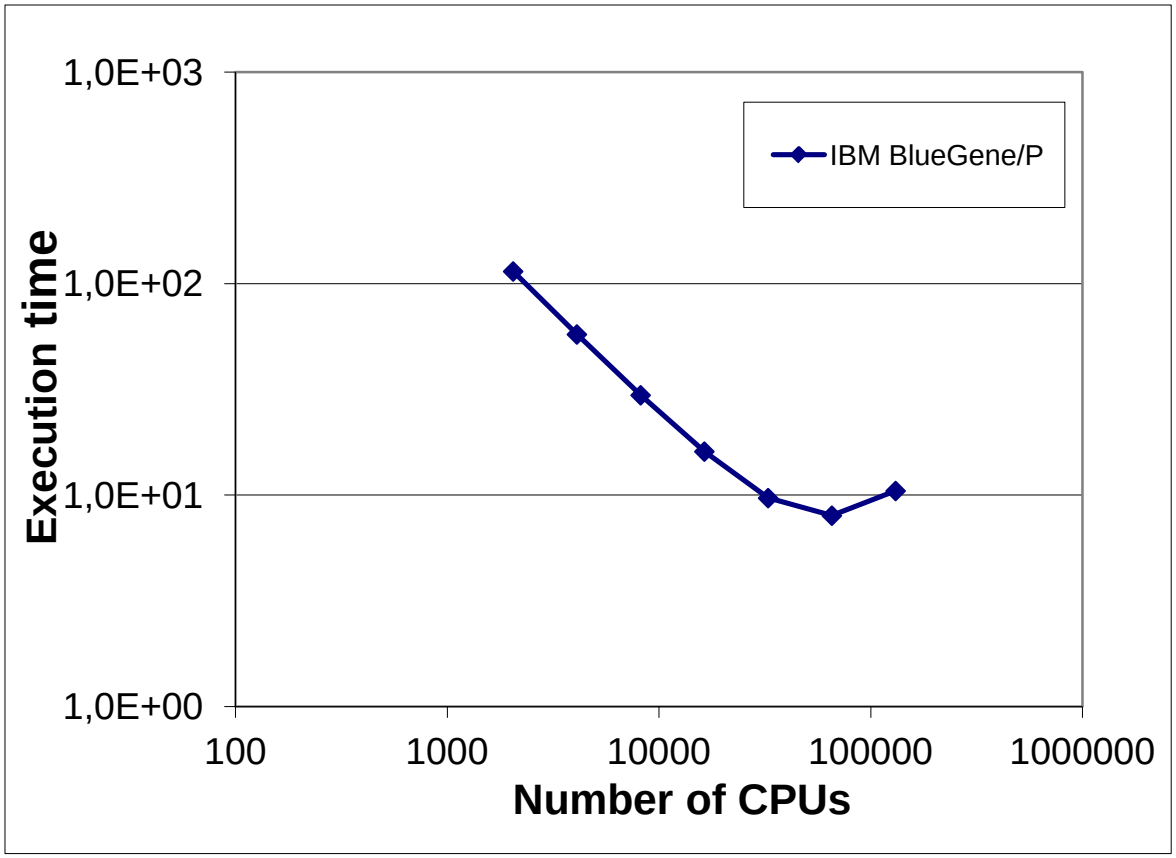

Figure 69: Execution time of PEPC, Test Case B

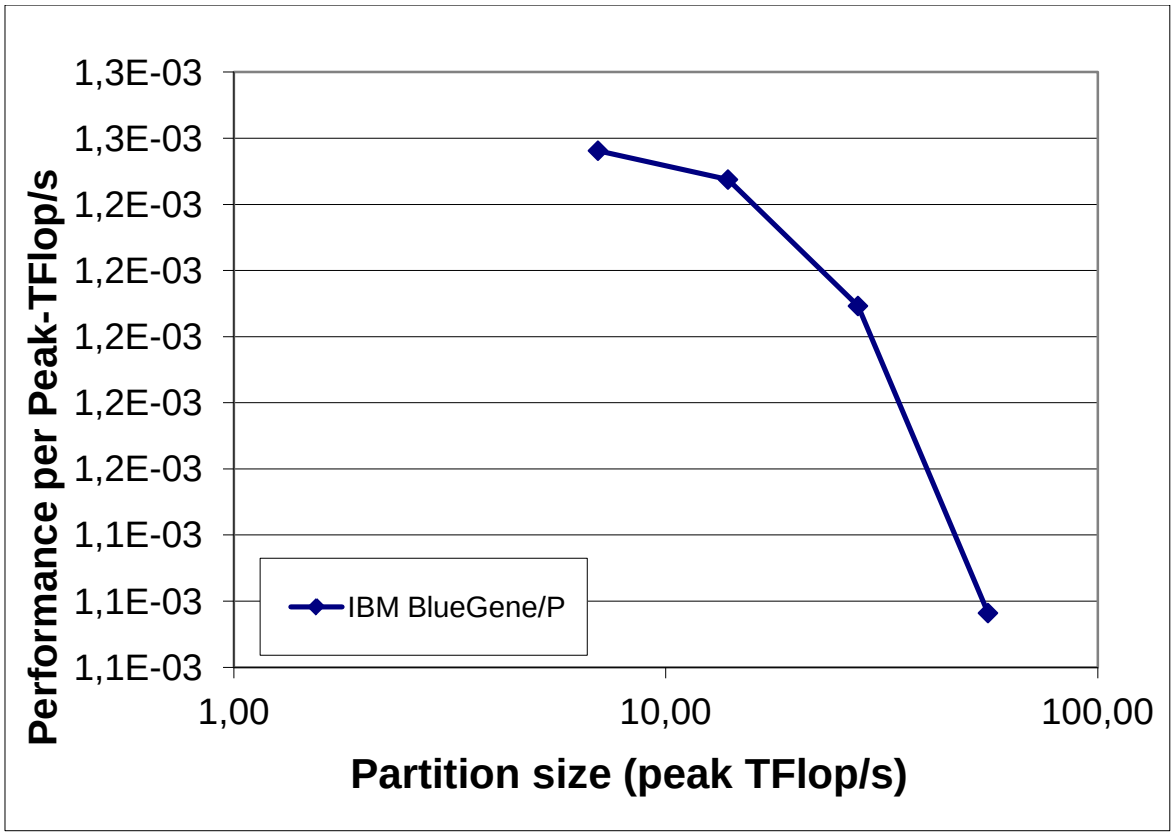

Figure 70: Performance per Peak-TFlop/s for PEPC, Test Case B

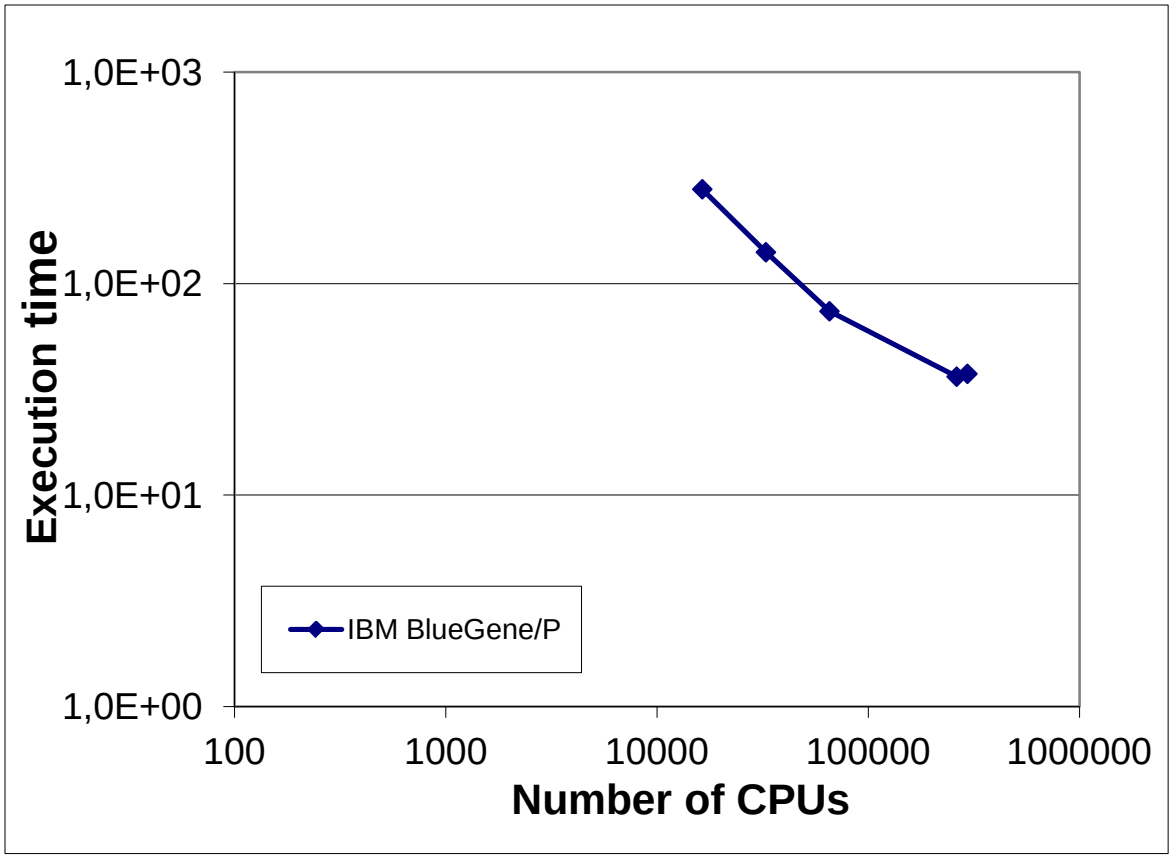

Figure 71: Execution time of PEPC, Test Case C

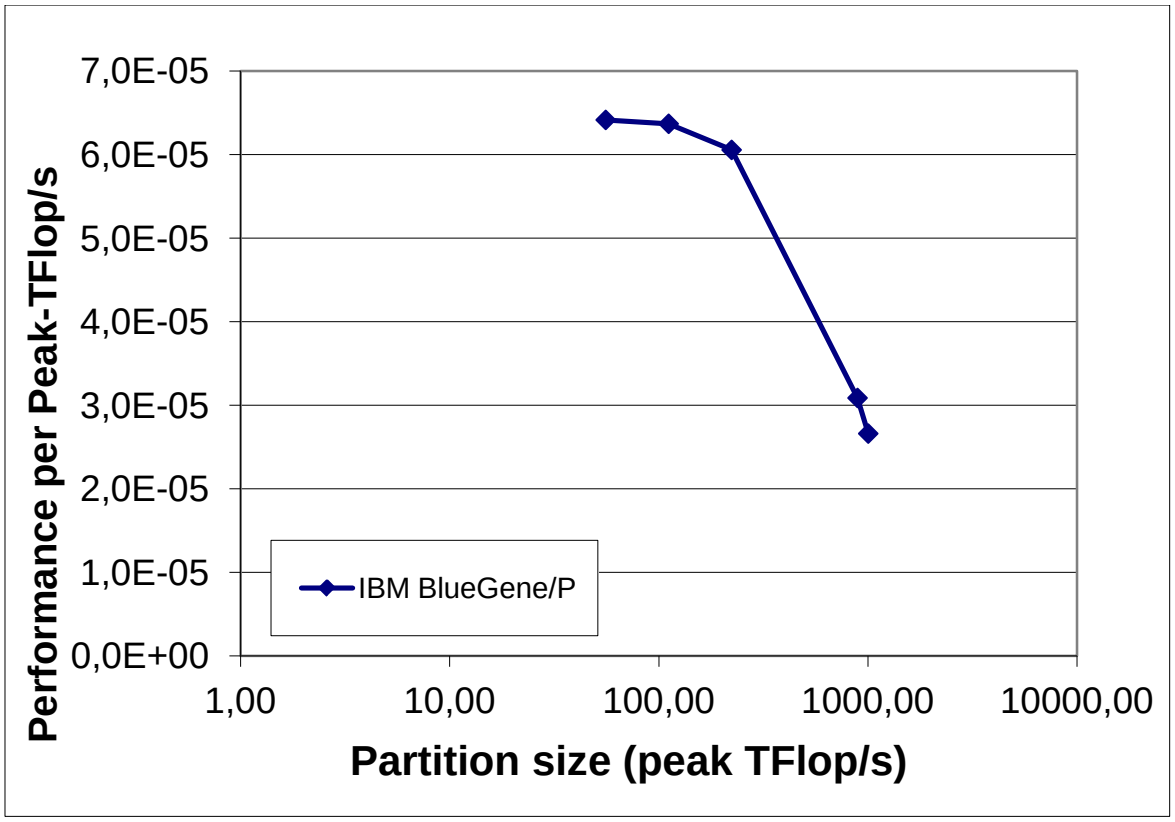

Figure 72: Performance per Peak-TFlop/s for PEPC, Test Case C

3.19.4 *Analysis*

PEPC could only be run on JUGENE because of problems on CURIE with the new threaded version and the currently available MPI. Therefore it is not possible to compare the results between the two systems, but it can be seen that the new code version scales well to high core numbers on JUGENE if a large set of particles is under consideration, looking at Test Case C it can be seen that nearly the full machine can be used efficiently.

3.20 SPECFEM3D

3.20.1 *Summary*

SPECFEM3D_GLOBE simulates global and regional (continental-scale) seismic wave propagation. Effects due to lateral variations in compressional-wave speed, shear-wave speed, density, a 3D crustal model, ellipticity, topography and bathymetry, the oceans, rotation, and self-gravitation are all included.

The 5.1 release offers non-blocking MPI communications and includes a significant improvement in performance and a more accurate implementation of the crust. The latest version provides a perfectly load-balanced mesh for 3D mantle models honoring shallow oceanic Moho (depths less than 15 km) and deep continental Moho (depths greater than 35 km). It also accommodates the European crustal models EPcrust (Molinari & Morelli, 2011) and EuCrust07 (Tesauro et al., 2008), which may be combined with the global crustal model Crust2.0. Sedimentary wavespeeds are superimposed on the mesh if sediment thickness exceeds 2 km.

3.20.2 *Test Cases*

Test Case A is a large test dimensioned in order to use efficiently the memory available on CURIE, running from 864 cores (mesh 576 12 / points 14693742139, degrees of freedom: 39865526769 - work amount ratio per core: 46140656) to 6144 cores (mesh 896 28 / points 95404911099, degrees of freedom: 260124839409 - work amount ratio per core: 42338027). Full populated nodes (with 32 cores per node) are used.

Test Case B is the same case using twice nodes – half populated version, so we get the boost of using underpopulated nodes.

3.20.3 *Results*

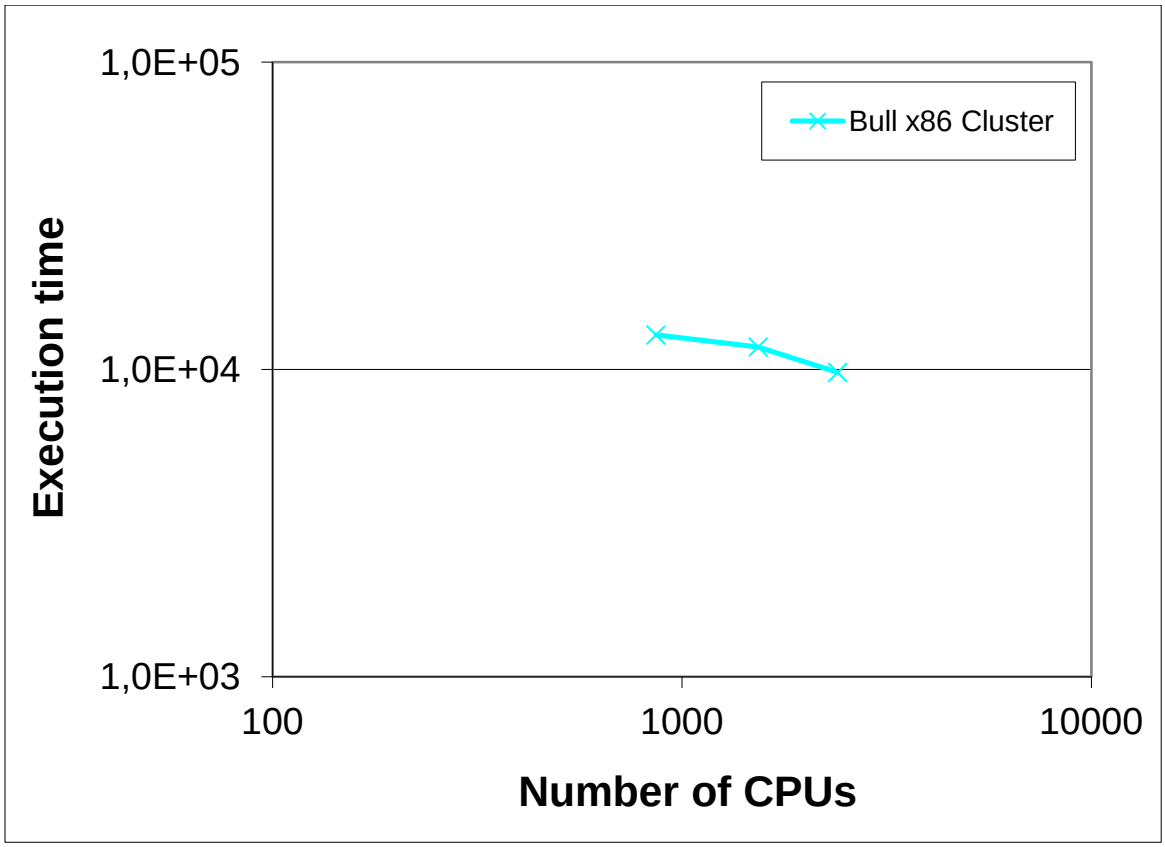

Figure 73: Execution time of SPECFEM3D, Test Case A

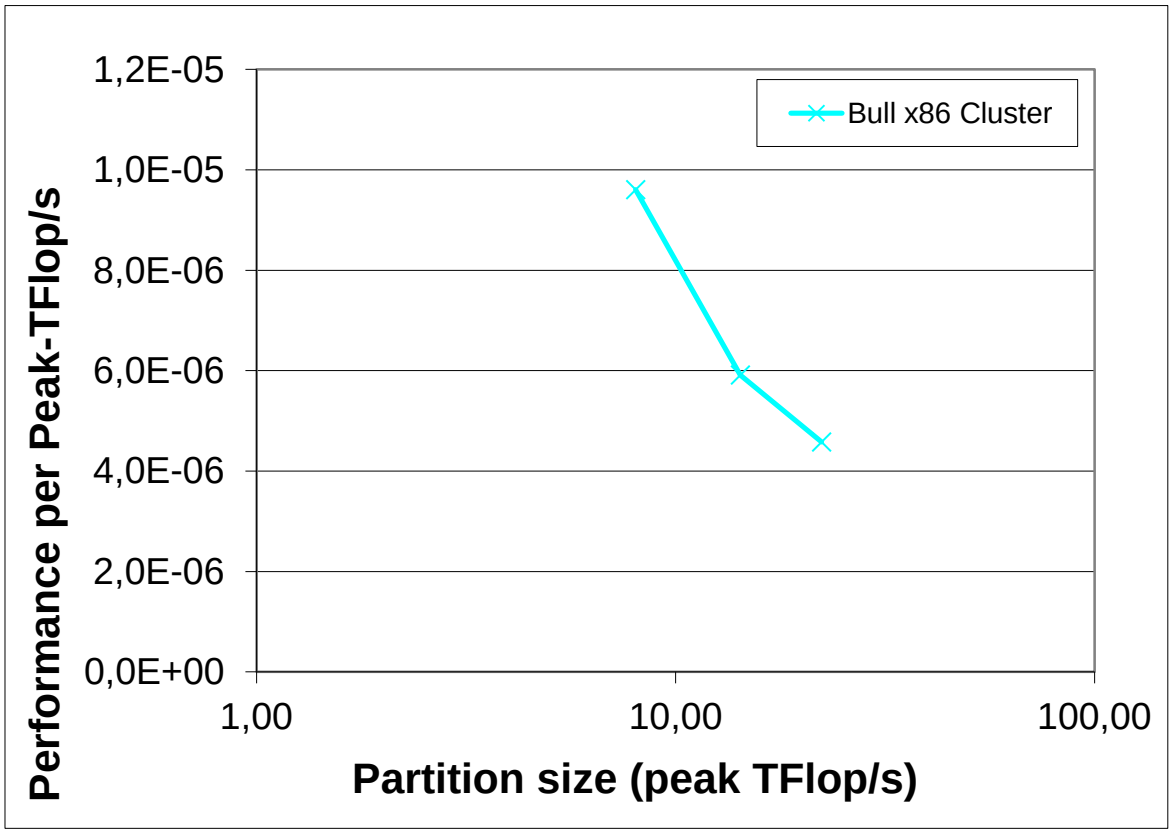

Figure 74: Performance per Peak-TFlop/s for SPECFEM3D, Test Case A

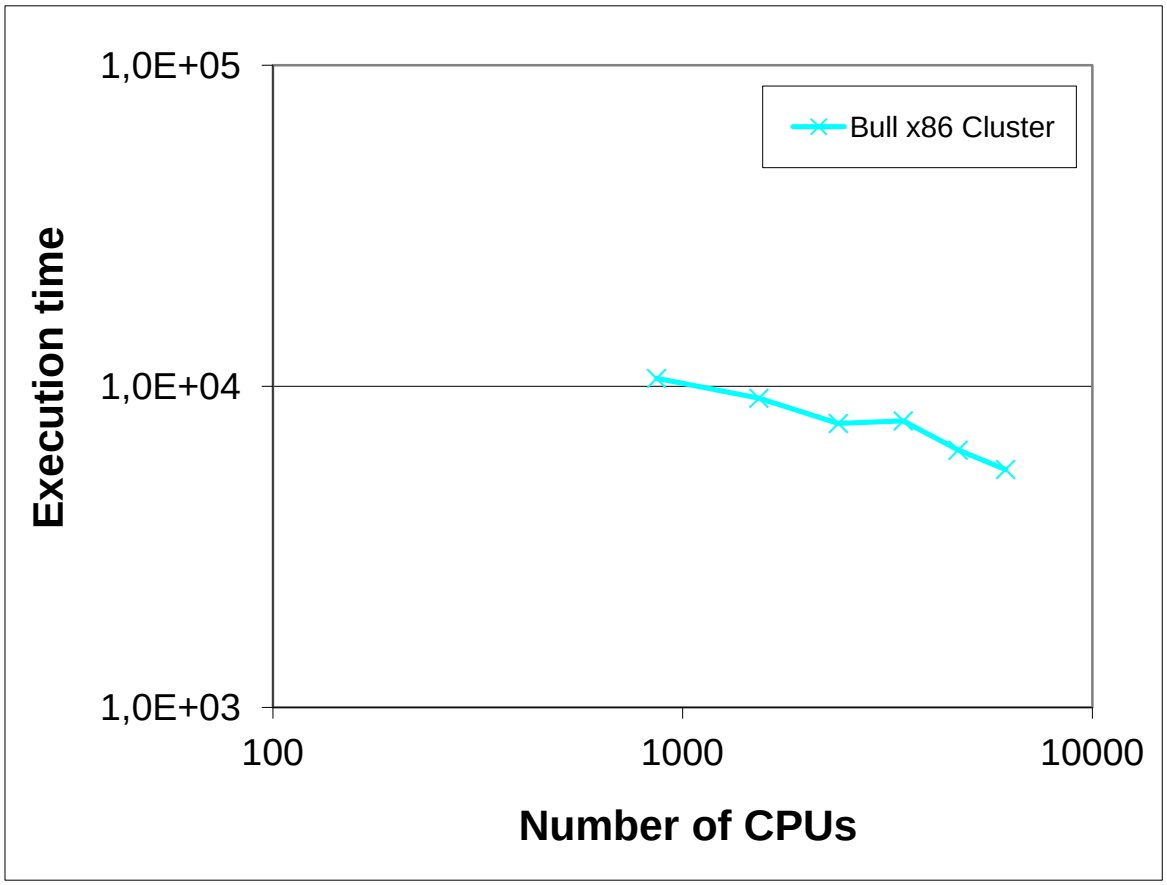

Figure 75: Execution time of SPECFEM3D, Test Case B

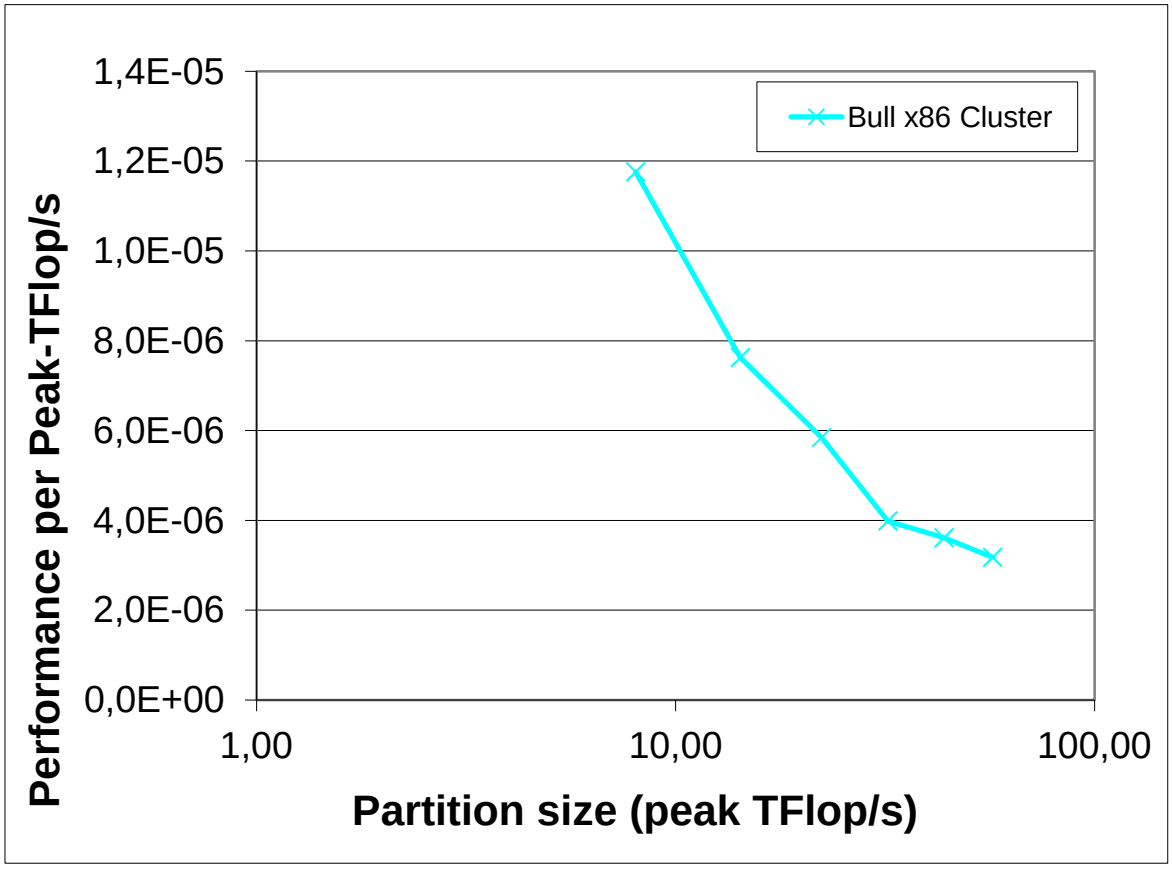

Figure 76: Performance per Peak-TFlop/s for SPECFEM3D, Test Case B

3.20.4 *Analysis*

As expected, on unpopulated nodes (test case B) a speed boost of about 20% to 30% is observed versus the same computation (test case A) on full populated nodes (half number of nodes). With respect to scalability, for the full populated case this is somewhat mitigate, but still good use up to more than 6000 cores (about 200 nodes) is observed.

3.21 TRIPOLI_4

3.21.1 *Summary*

TRIPOLI_4 is a 3D general purpose Monte Carlo code to calculate neutrons/gamma flux related quantities: reaction rates, deposited energy, kerma etc., specially designed for deep penetration problems (radiation protection and shielding). It is used as a reference tool by CEA as well as its industrial or institutional partners, and in the NURESIM [2] European project. It is available from the NEA and RSICC web sites.

3.21.2 *Test Cases*

Test Case A is a benchmark case (FNG Bulkshield) for reproducing 14 Mev neutron transport through ITER radiological protections (neutron source defined by FZK, geometry by SERMA using IPP's MCAM software). Target value to be computed by the code is the estimated Mn55 detector value with its variance (without/with reduction algorithms).

As computation can continue as long as is wished by the user, with an improved result that variance get smaller and smaller (diminishing in 1/sqrt(n), n being number of processed neutrons during computation), the relevant performance indicator of the computation is obtained by Perf=1/(Telapsed_to_thispoint*variance_to_this_point**2).

As a way of normalising results to be coherent to other codes, for execution time an equivalent time has been used, computed by Tequiv=Telapsed_to_this_point*Treference/Perf where Treference is arbitrary (it was set it in order to get the same value for "to_this_point" and "equivalent" times for the lowest CPU point – this is equivalent to normalising to the variance of this 1st computation, that is stopping the computation for any number of CPU when we get identical variance as 1st one).

3.21.3 *Results*

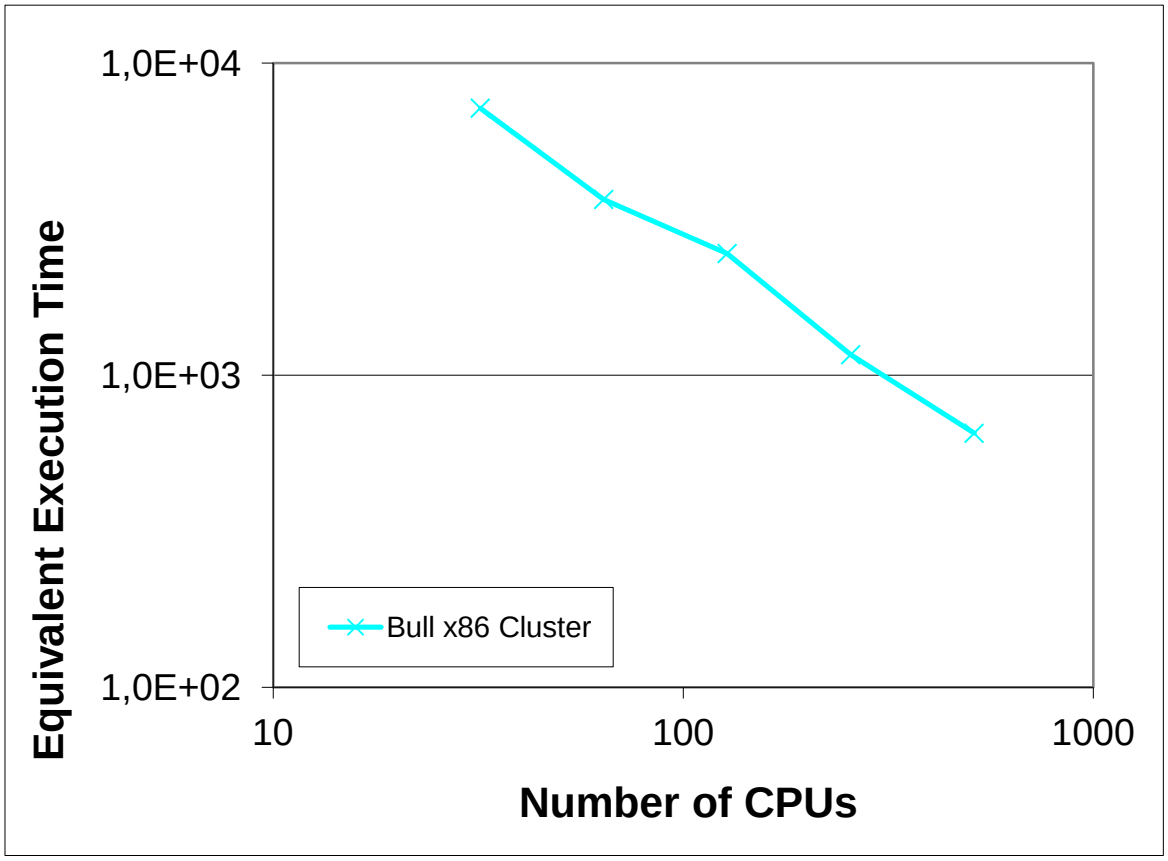

Figure 77: Equivalent execution time for TRIPOLI_4, Test Case A

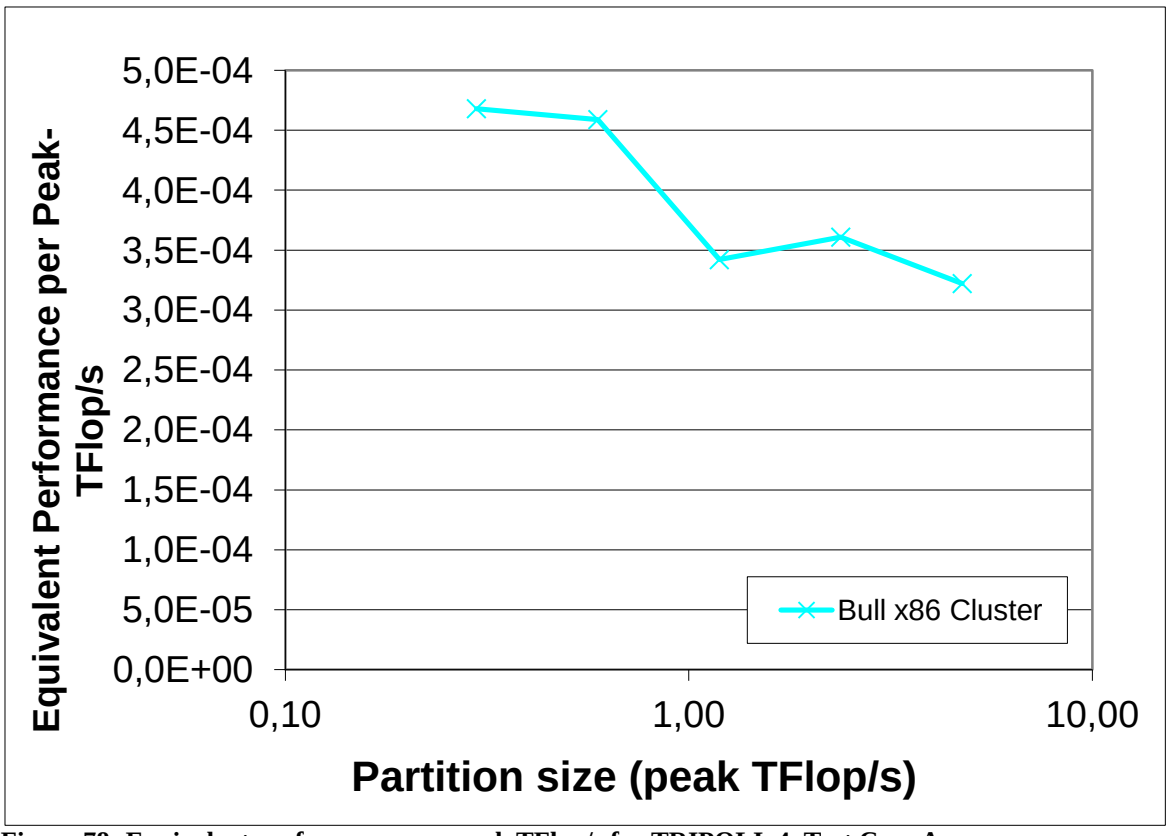

Figure 78: Equivalent performance per peak TFlop/s for TRIPOLI_4, Test Case A

3.21.4 *Analysis*

For the two first measurement points we get “intra node performance” as the number of task is less then the core number for the node, and so we see the associated performance boost of intra node communications (about +33%). For later points we get a very good scalability (essentially constant performance) as the code is embarrassingly parallel - efficiency for 512 cores is still about 94% versus the smallest multimode value on two nodes = 128 cores).

3.22 Application Benchmarking Summary

In the Preparatory Project all codes were compared to runs on a reference platform. However, this was not possible in PRACE-1IP because there is no platform from which we have results for all codes. In Figures 79 and 80, the performance per Peak-TFlop/s is normalised to the performance on a 10 TFlop/s partition. Note, that not all codes can be shown, because for some we could not calculate a value for the 10 TFlop/s partition. These graphs give an overall impression of how well the applications scale on the two systems. In general we see that the curves are flatter on the IBM BlueGene/P than on the Bull x86 Cluster, which indicates that scalability is, on the whole, better on the former system. For NAMD and GADGET, however, scalability on IBM BlueGene/P is poor, as indicated by the steeply declining curves. For these codes the Bull x86 Cluster seems to be more appropriate.

For most of the applications where we are able to make the comparison, the Bull x86 Cluster often offers better performance per Peak-Tflop/s than the IBM BlueGene/P, at least for small partition sizes (notable exceptions are ALYA and Quantum_Espresso).

For larger partition sizes however, the poorer scalability on the Bull x86 Cluster means that this performance gap tends to decrease and in some cases the IBM BlueGene/P becomes the more efficient solution. We would expect this trend to be more evident if we were able to run on a larger Bull x86 Cluster system.

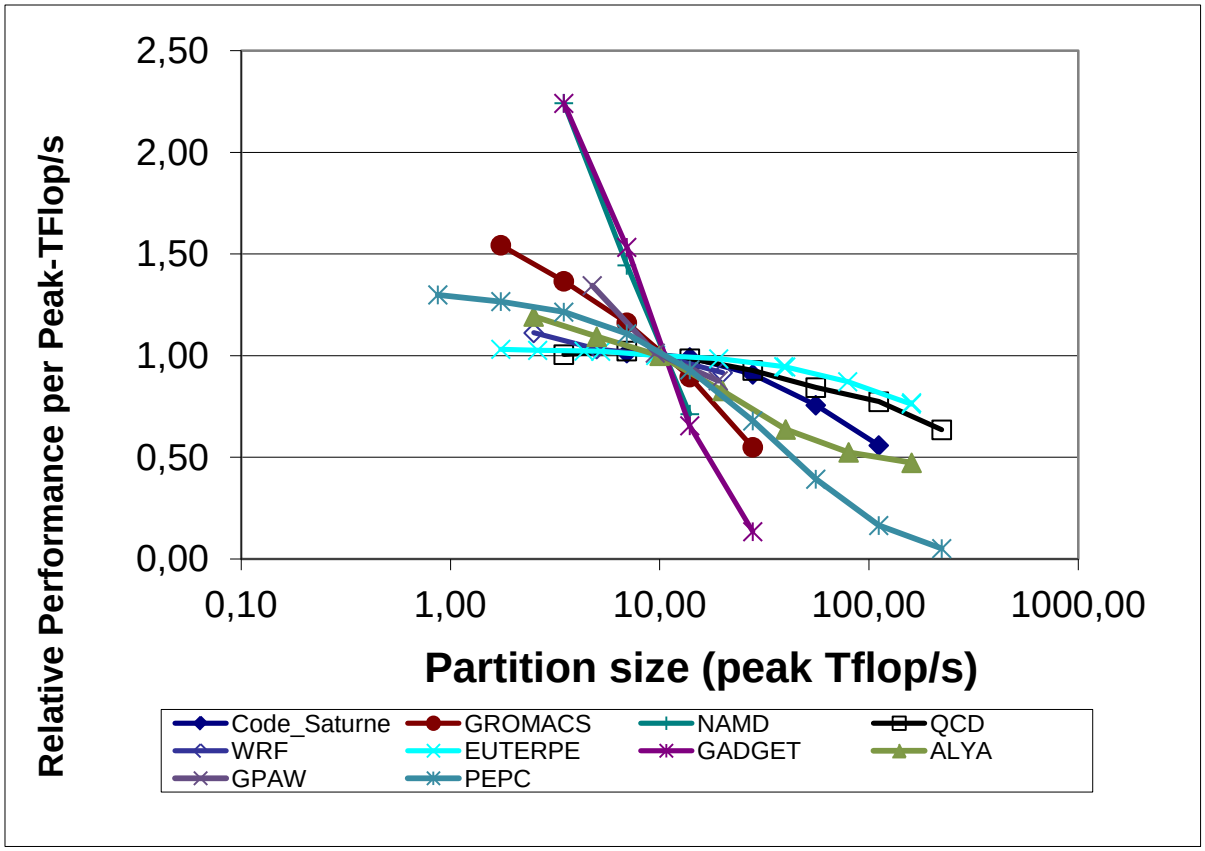

Figure 79: Relative performance per Peak-TFlop/s of applications on IBM BlueGene/P

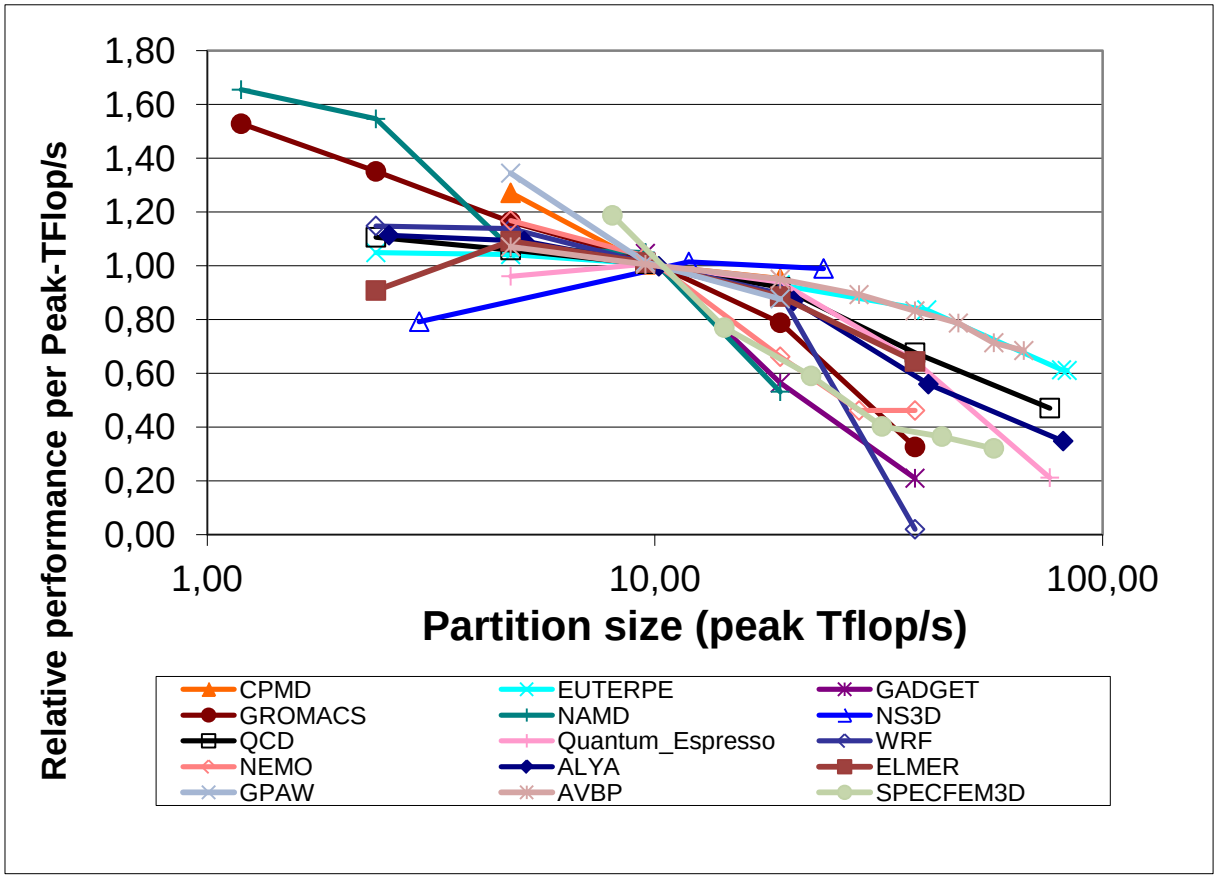

Figure 80: Relative performance per Peak-TFlop/s of applications on Bull x86 Cluster

4 Synthetic Benchmarking

Another objective of Task 7.4 was to analyse the synthetic benchmarks used by the PRACE Preparatory Phase on the Tier-0 systems[1]. These benchmarks help in particular to understand the system behaviour and to relate it to user applications in order to assist them to assign a suitable system for their needs. Because these benchmarks were run on JUGENE in the Preparatory Phase, and the system has not changed significantly since then, the runs were not repeated on JUGENE. The results for JUGENE can be found in [5] and [6].

4.1 Synthetic Benchmarks Results

The synthetic benchmarks have been performed on the new Tier-0 systems CURIE. The results are presented here, in the same order, and following the same recipes used in the Preparatory Phase and as described in [5].

4.1.1 *T1.1 LINPACK Sustained Flop/s*

This is a measure of sustained double precision (DP) floating-point operations per second (flop/s) using the LINPACK benchmark which solves a dense linear system of equations. It is used to provide a flop calculation rate measure close to peak performance. The benchmark is included here because of the popularity of LINPACK in ranking floating-point performance in the TOP500 list. In total 9 jobs have been run, combining 1 up till 32 nodes. The highest measured rate was 7829.00 Gflop/s on 32 nodes with 1024 cores. The measured data points form an almost perfect straight line.

General information

Compiler options:	-O3 -ftz -ip -ipo
Run on:	4-12-2011 and 7-12-2011
Run by:	Soon-Heum „Jeff“ Ko

Results

nodes/cores	Input Parameters				Gflop/s
	N	NB	P	Q	
1/1	20480	128	1	1	8.22
1/4	40960	128	2	2	32.20
1/16	81920	128	4	4	127.30
2/32	115712	128	4	8	247.40
2/64	163840	128	8	8	505.30
4/128	231680	128	8	16	1018.00
8/256	327680	128	16	16	2006.00
16/512	463360	128	16	32	3998.00
32/1024	655360	128	32	32	7829.00

Table 1 CURIE at TGCC - LINPACK sustained Gflop/s for various numbers of nodes

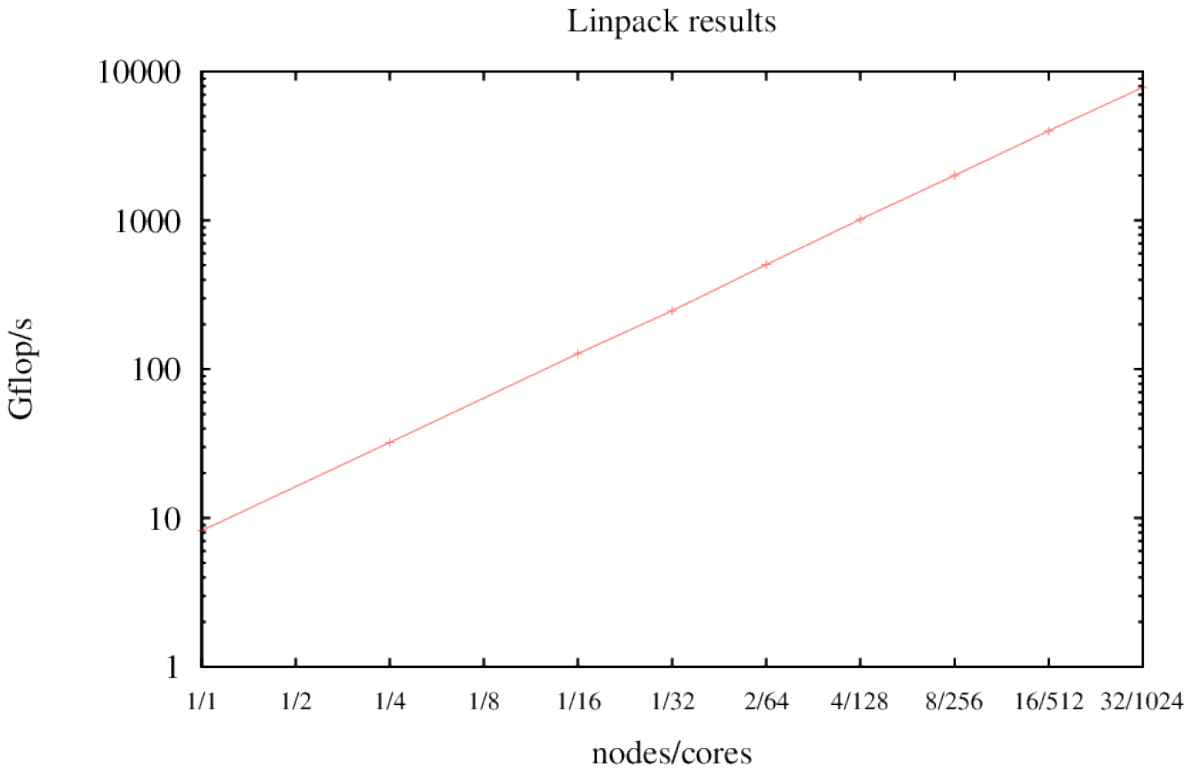

Figure 81: CURIE at TGCC – LINPACK sustained Flop/s

4.1.2 *T1.2 EuroBen Intrinsic Operations*

The first set of tests in the EuroBen benchmark suite provides a number of tests for intrinsic operations from cache and main memory and report flop/s and operations per second. The shared memory EuroBen version is run on an increasing number of cores on a node to measure scaling effects.

For the prototype assessments we have selected the following 3 modules (benchmark programs):

- mod1a - basic operations from cache (if present);
- mod1b - basic operations from main memory;
- mod1f - intrinsic mathematical functions.

Each benchmark module contains a large number of different operations and so the results presented here are a selection of these. EuroBen intrinsic operations benchmarks mod1a, mod1b, and mod1f have been run for an increasing number of cores within one node, viz. from 1 to 32.

General information

Compiler options:	-O3 -ftz -ip -ipo
Run on:	30-11-2011
Run by:	Jeroen Engelberts

Results

Tasks	Kernel 5	Kernel 6	Kernel 7	Kernel 8	Kernel 14
1	805.15	630.12	2518.90	2227.20	3716.70
2	1594.90	1127.40	4090.00	4090.00	7174.20
4	2512.60	1904.80	5865.10	5747.10	13514.00
8	3300.30	2659.60	6622.50	7326.00	23166.00
16	363.37	349.90	674.31	707.21	20930.00
32	1142.90	1113.60	777.00	2012.10	17664.00

Table 2 CURIE at TGCC - EuroBen shared memory mod1a (MFlop/s)

Tasks	Kernel 5	Kernel 6	Kernel 7	Kernel 8	Kernel 14
1	208.08	207.82	596.28	428.25	1822.40
2	262.39	320.19	1155.50	968.74	3515.80
4	534.82	407.48	1295.00	1233.60	6362.50
8	647.01	655.48	1997.00	1606.60	12173.00
16	482.57	486.49	1553.20	1090.50	9807.90
32	530.97	526.59	2225.20	1576.70	14561.00

Table 3 CURIE at TGCC - EuroBen shared memory mod1b (MFlop/s)

Function	1 core	2 cores	4 cores	8 cores	16 cores	32 cores
x**y	29.07	57.80	113.64	217.39	392.16	243.90
Sin	116.28	224.72	454.55	833.33	952.38	833.33
Cos	106.38	210.53	416.67	769.23	689.66	800.00
Sqrt	141.84	277.78	540.54	1000.00	952.38	869.57
Exp	162.60	322.58	625.00	1176.47	952.38	869.57
Log10	99.01	178.57	377.36	740.74	689.66	800.00
Tan	111.73	217.39	434.78	800.00	909.09	833.33
Cot	99.50	194.18	377.36	714.27	740.74	800.00
Asin	85.47	163.93	333.33	625.00	869.57	769.23
Acos	91.32	175.44	338.98	689.66	952.38	800.00
Atan	75.19	147.06	294.12	571.43	869.57	800.00

Table 4 CURIE at TGCC - EuroBen shared memory mod1f (MFlop/s)

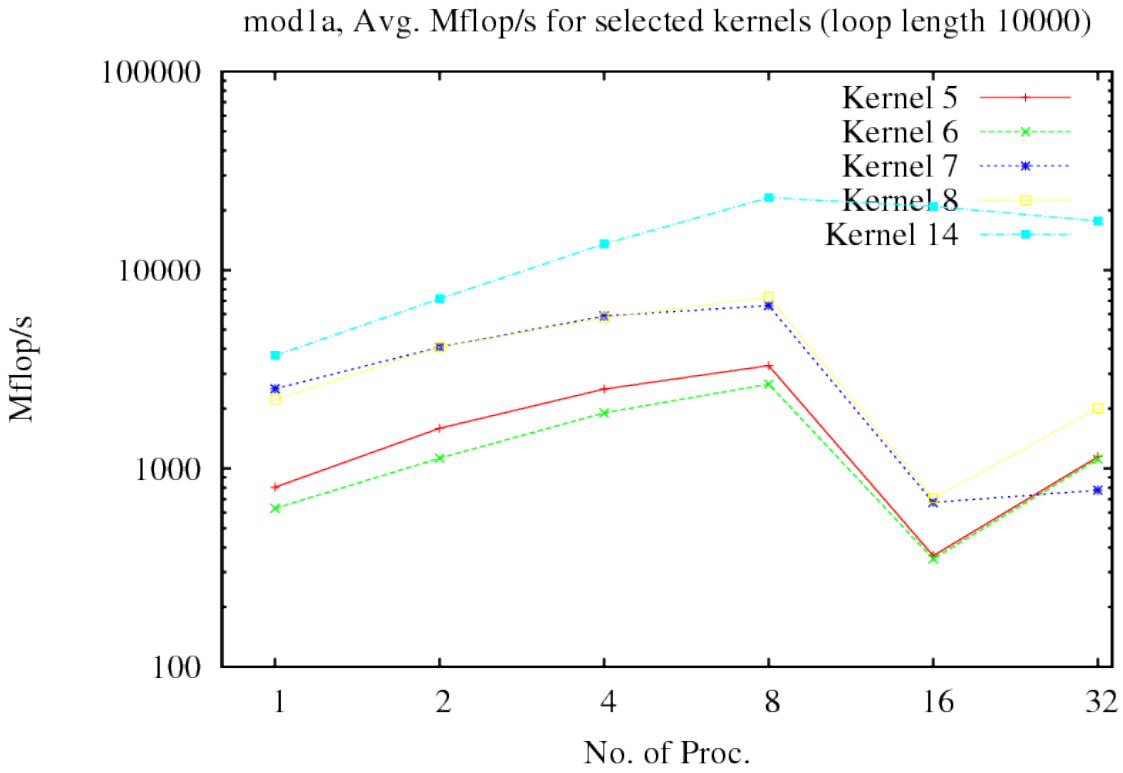

Figure 82: CURIE at TGCC - EuroBen shared memory mod1a

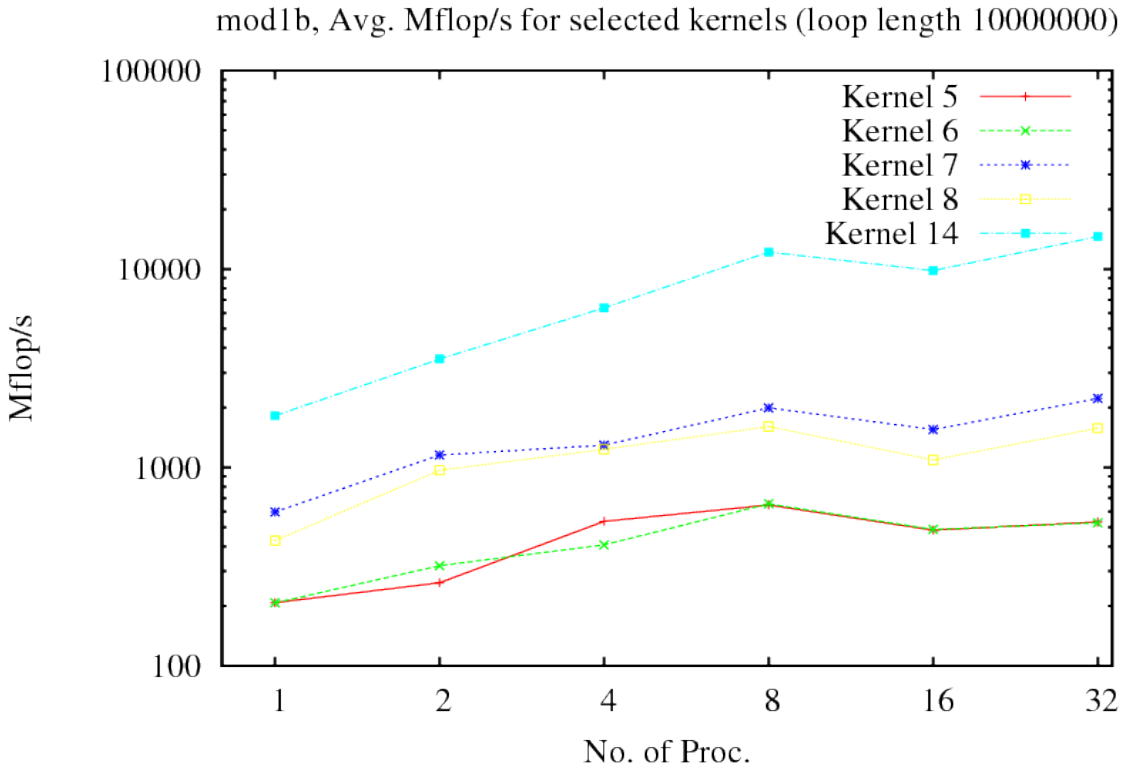

Figure 83: CURIE at TGCC - EuroBen shared memory mod1b

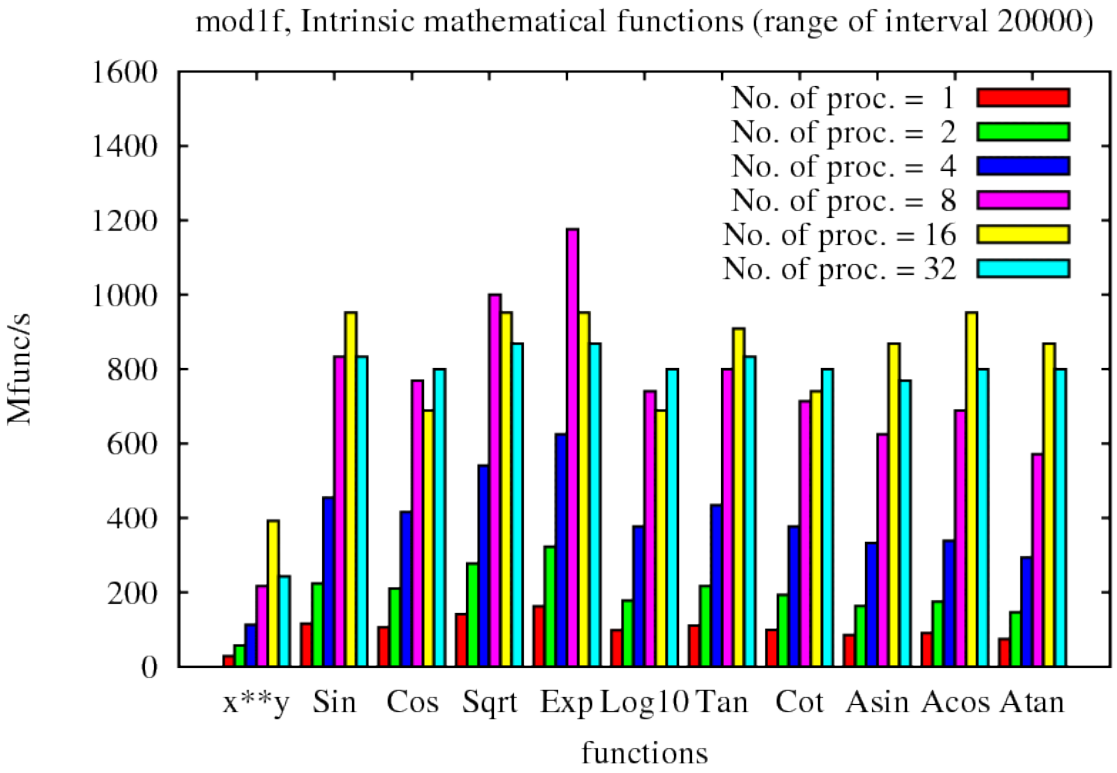

Figure 84: CURIE at TGCC - EuroBen shared memory mod1f

4.1.3 *T1.3 EuroBen Representative Algorithms*

The second set of tests in the EuroBen benchmark suite implements a number of representative algorithms. For all the tests, except for mod2d, the MPI version of EuroBen is used to test scaling over multiple nodes. The mod2d test is only available as a shared memory benchmark and this is scaled over cores on a single node.

For this assessments the following 8 modules (benchmark programs) have been selected:

- mod2a - vector update **c** = **c** + **Ab**;
- mod2as - vector update **c** = **c** + **Ab** for sparse matrix A is CRS format;
- mod2ci - solving a sparse linear non-symmetric system **Ax** = **b** of Finite Element type. Matrix is in CRS format;
- mod2cr - solving a sparse linear symmetric system **Ax** = **b** stemming from a 3-D finite difference problem;
- mod2d - finding eigenvalues of a full Real and symmetric matrix **A**;
- mod2g - 2-D Haar Wavelet Transform;
- mod2h - random number generator;
- mod2i - sorting Integers and 64-bit Reals.

The EuroBen representative algorithm benchmark mod2d was scaled within a single host. The other benchmarks in this section were scaled within one node up to and including 32 tasks andsubsequently over multiple nodes each running 32 tasks. Some benchmarks failed to produce valid output at some level of scaling. Furthermore, it proved not to be possible to run mod2as on more than 2 nodes, or (2 x 32) 64 cores. Also mod2cr on 1024 processors failed to produce valid output, crashing with a segmentation fault.

Mod2g and mod2i could not be scaled up beyond 16 cores within a single node. With 32 cores within one node and multiple nodes, mod2g and mod2i crash with a segmentation fault.

General information

Compiler options:	-O3 -ftz -ip -ipo
Run on:	30-11-2011
Run by:	Jeroen Engelberts

Results

Cores	mod2a	mod2as	mod2ci	mod2cr
1	154.90	570.70	1105.30	440.21
2	318.00	1128.90	2365.30	440.55
16	2544.00	5909.60	7435.70	1430.80
32	4971.00	14072.00	7456.50	2208.30
64	10070.00	26735.00	1388.10	2858.80
128	19890.00		969.86	4657.60
256	39510.00		843.94	7593.00
512	78270.00		743.73	14113.00
1024	146900.00		832.03	

Table 5 CURIE at TGCC - EuroBen distributed memory - mod2a, mod2as, mod2ci and mod2cr

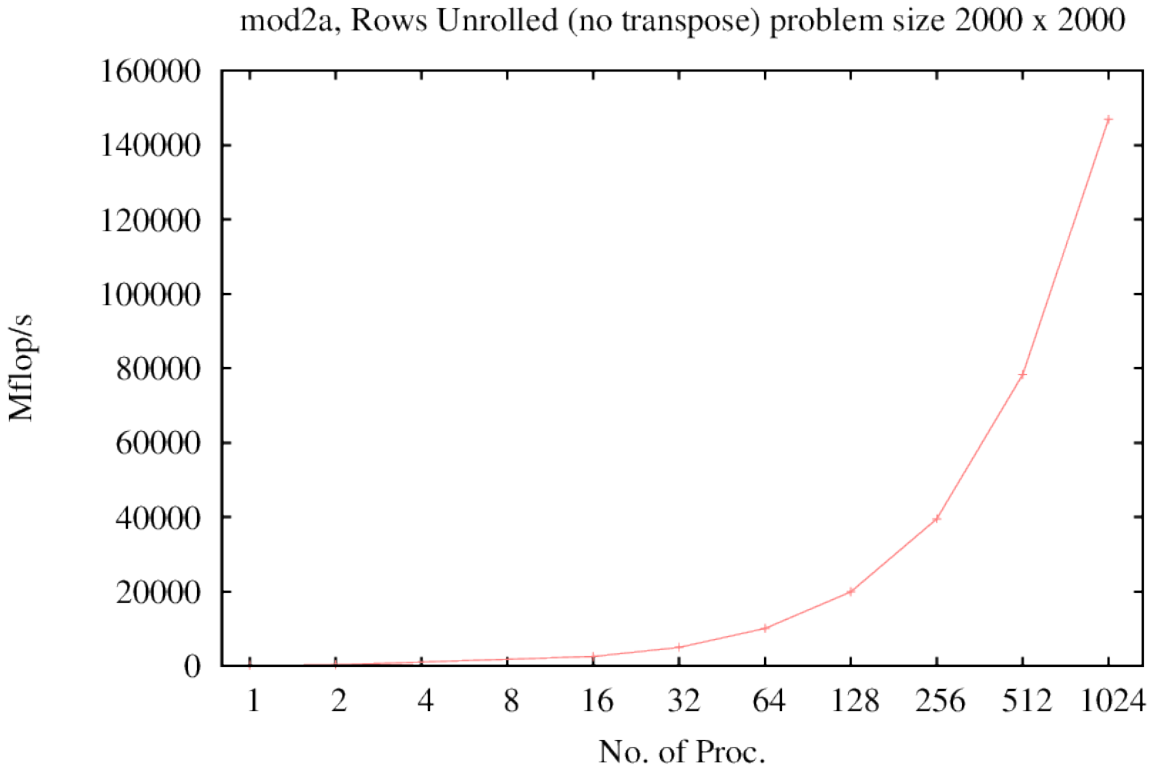

Figure 85: CURIE at TGCC - EuroBen mod2a

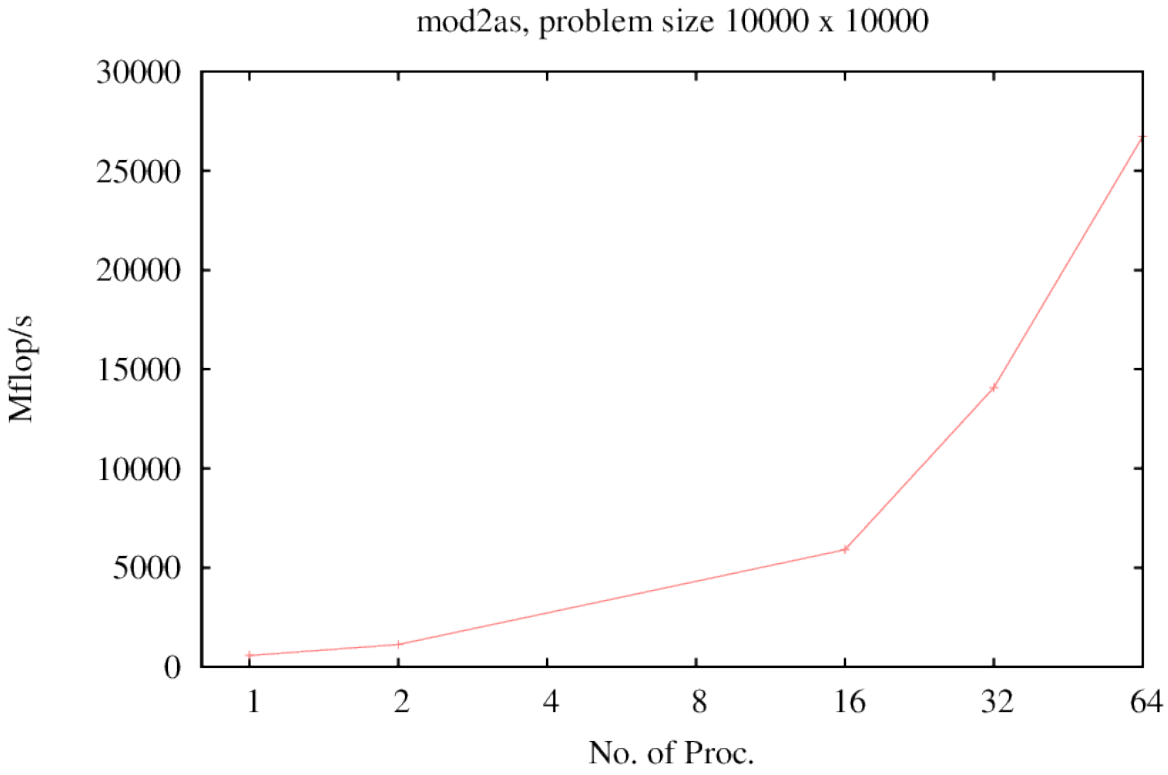

Figure 86: CURIE at TGCC – EuroBen mod2as

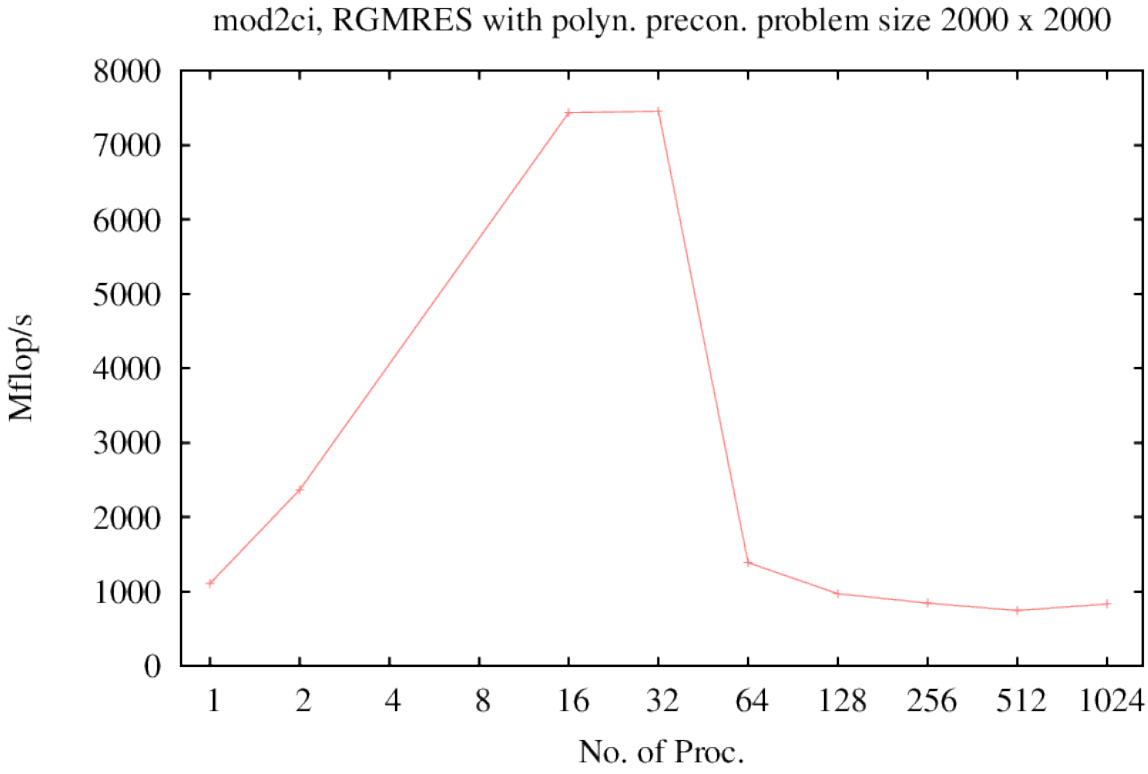

Figure 87: CURIE at TGCC - EuroBen mod2ci

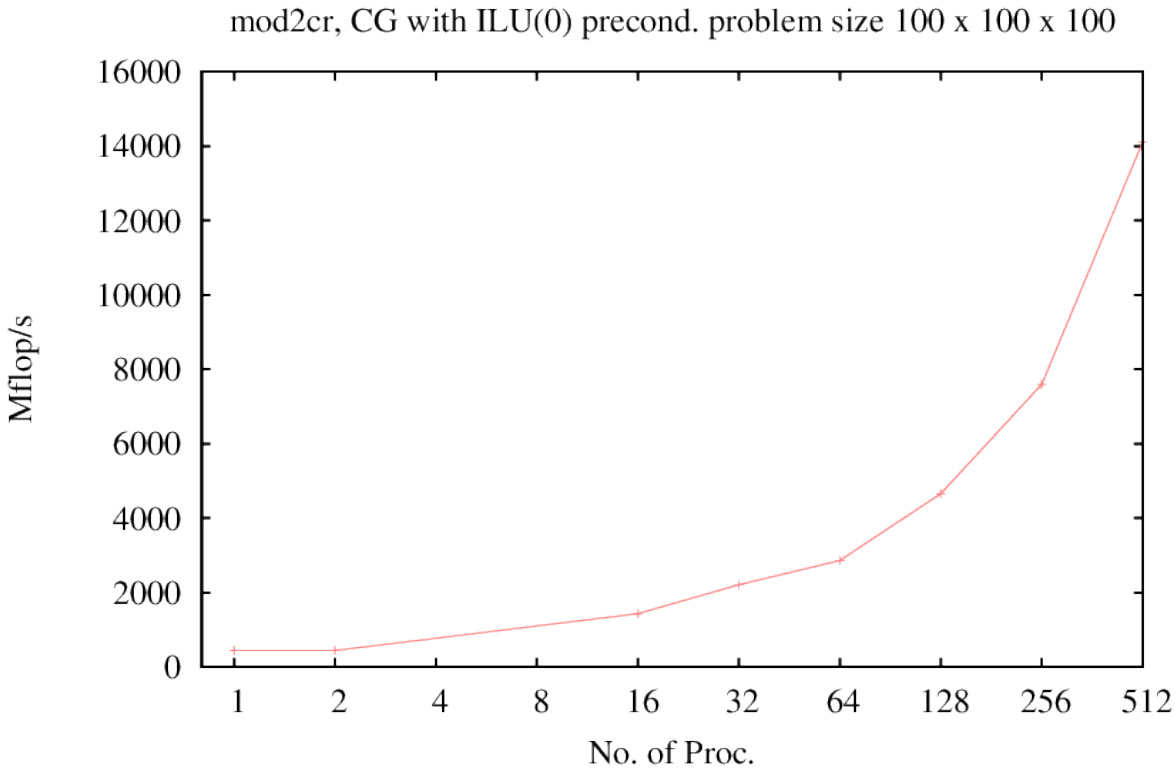

Figure 88: CURIE at TGCC - EuroBen mod2cr

Cores	mod2g	mod2h
1	575.03	547.17
2	997.15	1078.90
16	2214.60	4210.70
32		3372.00
64		1999.20
128		1021.60
256		521.99
512		261.88
1024		132.91

Table 6 CURIE at TGCC - EuroBen distributed memory - mod2g and mod2h

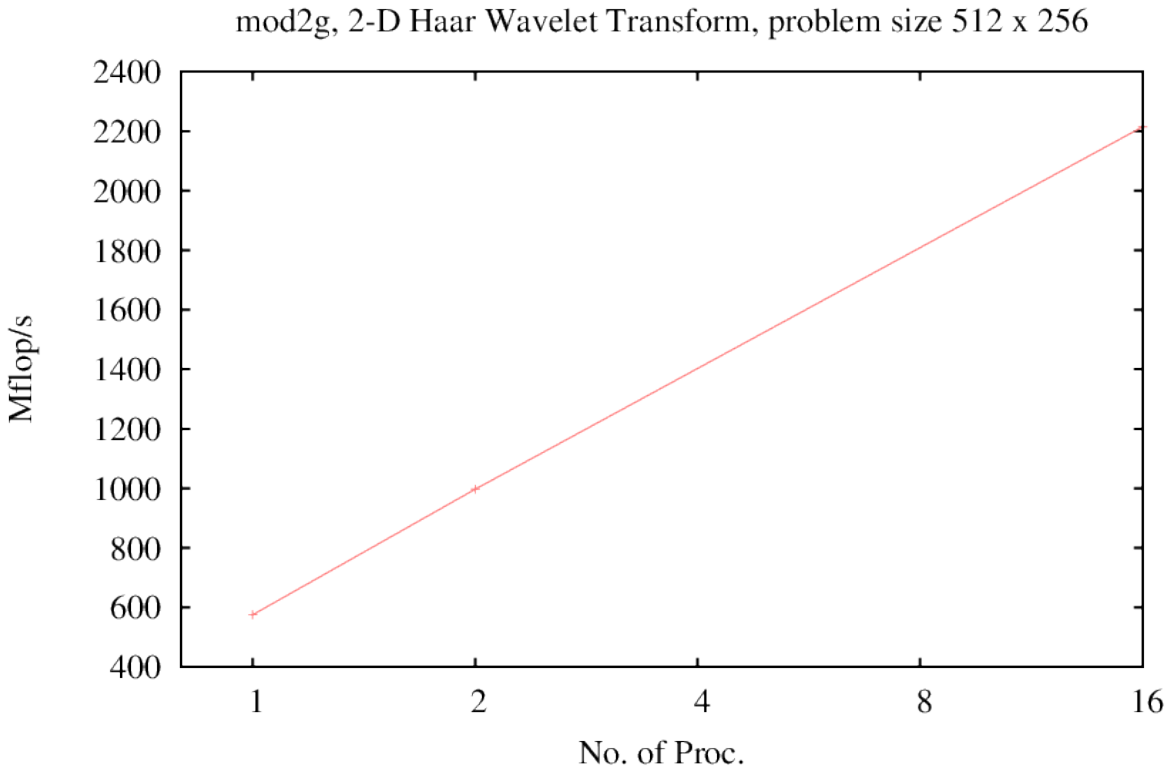

Figure 89: CURIE at TGCC - EuroBen mod2g

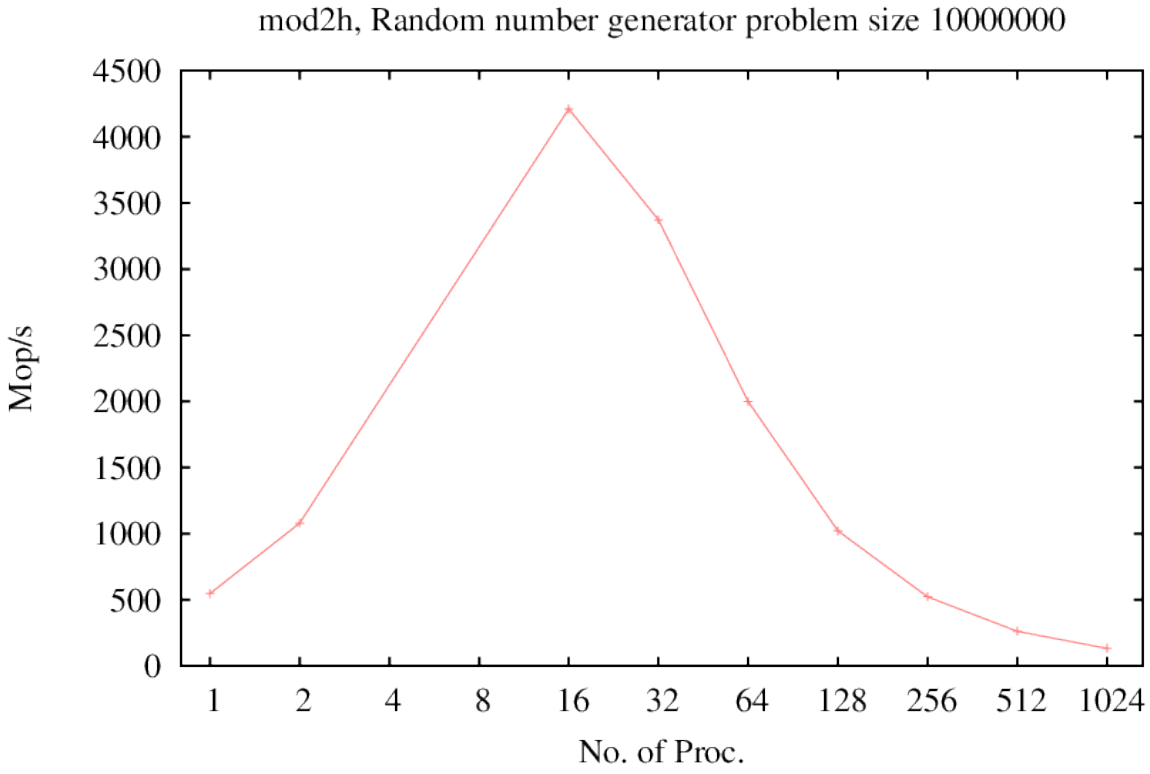

Figure 90: CURIE at TGCC - EuroBen mod2h

Cores	Integer	Double
1	161.30	128.30
2	295.40	231.90
16	2128.00	1356.00

Table 7 CURIE at TGCC - EuroBen distributed memory - mod2i

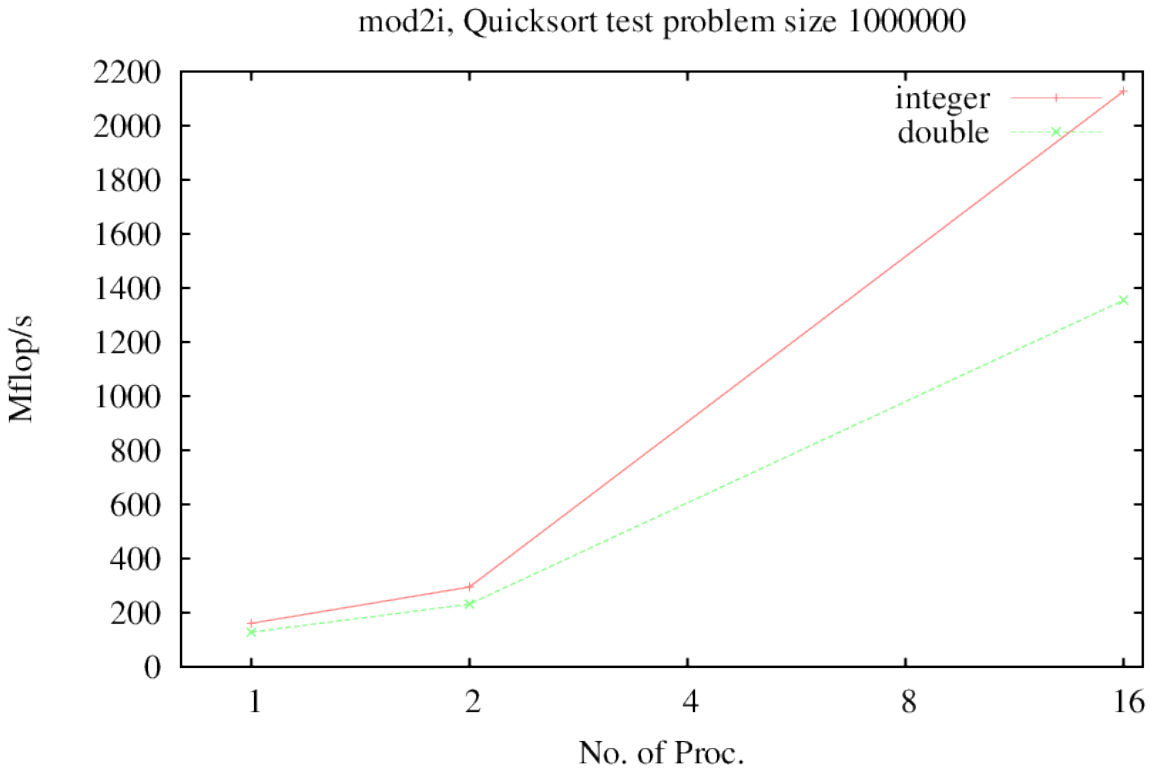

Figure 91: CURIE at TGCC - EuroBen mod2i

4.1.4 *T1.4 Sustained Memory Bandwidth*

This is a measure of sustained memory bandwidth using the Stream benchmark from HPCC. The purpose of this benchmark is to stress main memory.

The bandwidth drops roughly a factor 2 going from one core to multiple cores. Adding more cores does not decrease the bandwidth less.

General information

Compiler options:	-O3 -ftz -ip –ipo
Run on:	18-12-2011, 19-12-2011, 20-12-2011
Run by:	Soon-Heum „Jeff“ Ko

Results

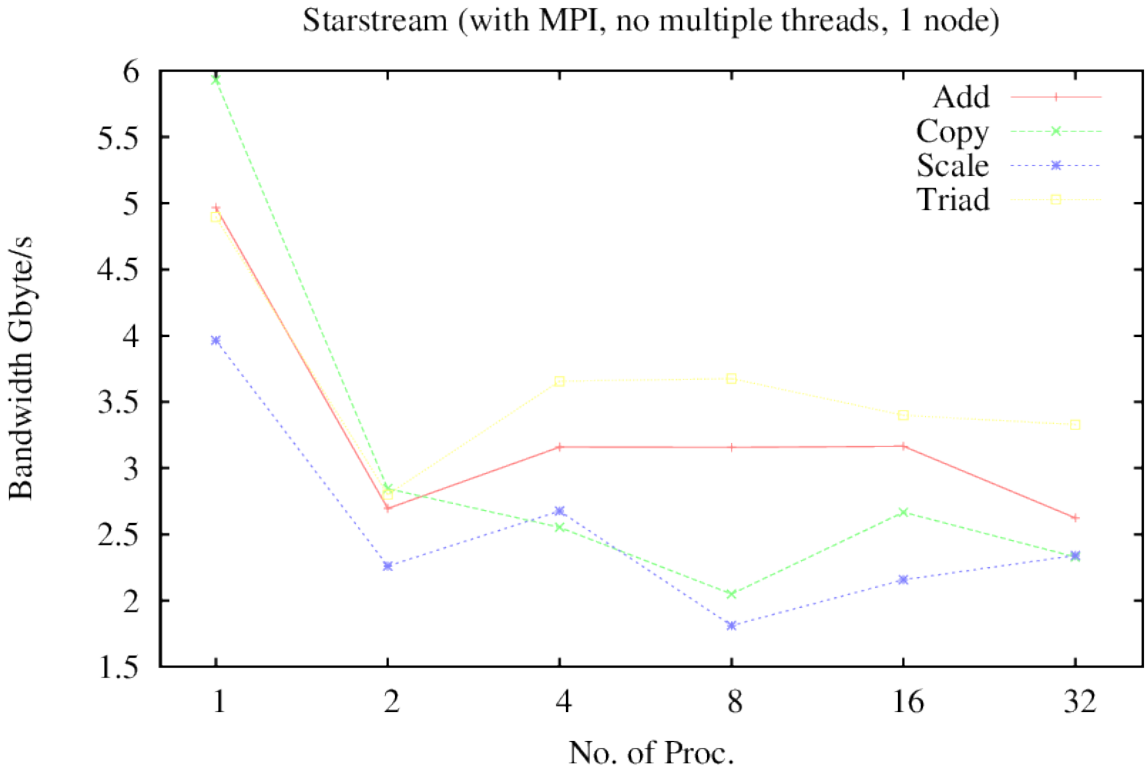

Figure 92: CURIE at TGCC - HPCC StarSTREAM per task sustained bandwidth

Cores	Add (GB/s)	Copy (GB/s)	Scale (GB/s)	Triad (GB/s)
1	4.9680	5.9307	3.9643	4.8958
2	2.6948	2.8451	2.2604	2.8002
4	3.1589	2.5527	2.6755	3.6559
8	3.1554	2.0491	1.8113	3.6758
16	3.1656	2.6675	2.1563	3.3987
32	2.6239	2.3287	2.3406	3.3284

Table 8 CURIE at TGCC - HPCC StarSTREAM

4.1.5 *T1.5 Sustained Memory Bandwidth at different Cache Levels*

STREAM2 is designed to measure sustained bandwidth at all levels of the cache hierarchy, and more clearly expose the performance differences between reads and writes [ref to STREAM2].

As the results below show, there is hardly any variation per operation for 1 – 32 cores for array sizes up to 300000 bytes. Above that array size the bandwidth drops around a factor 2 for 32 cores compared to 1 core.

General information

Compiler options:	-O3 -ftz -ip -ipo
Run on:	8-12-2011, 9-12-2011, 10-12-2011
Run by:	Carlos J. V. Simões

Results

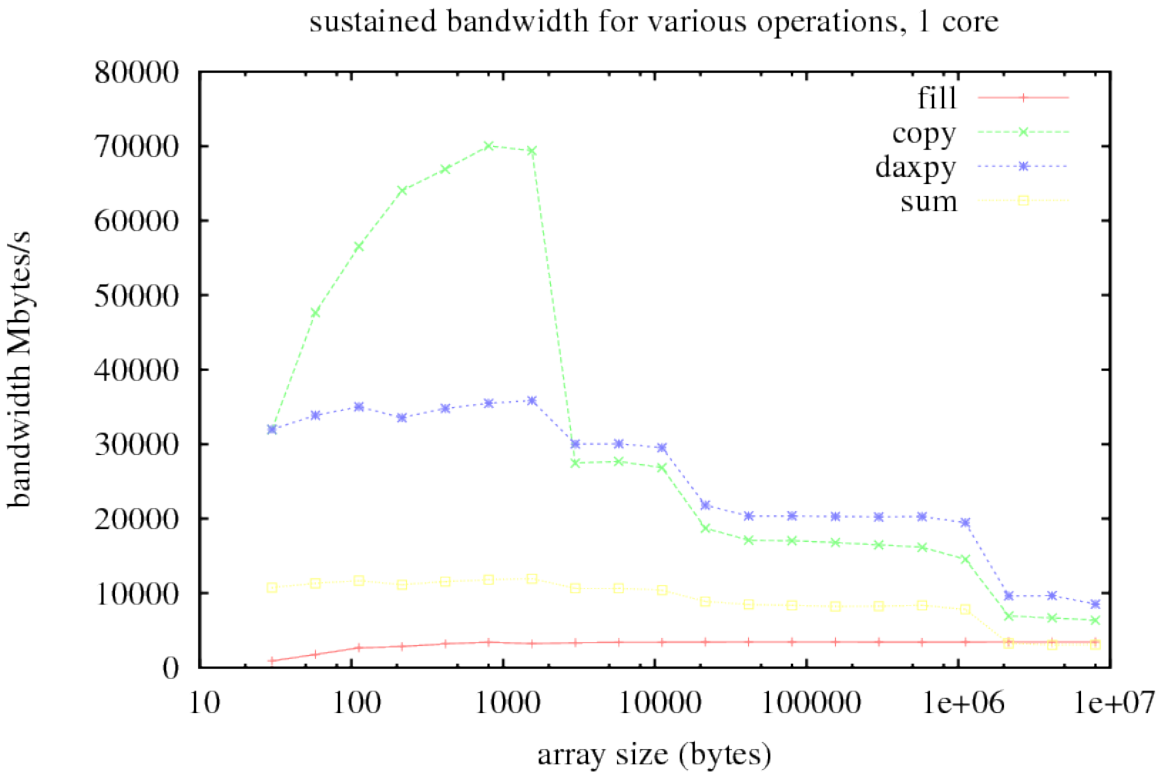

Figure 93: CURIE at TGCC - HPCC stream2 1 core

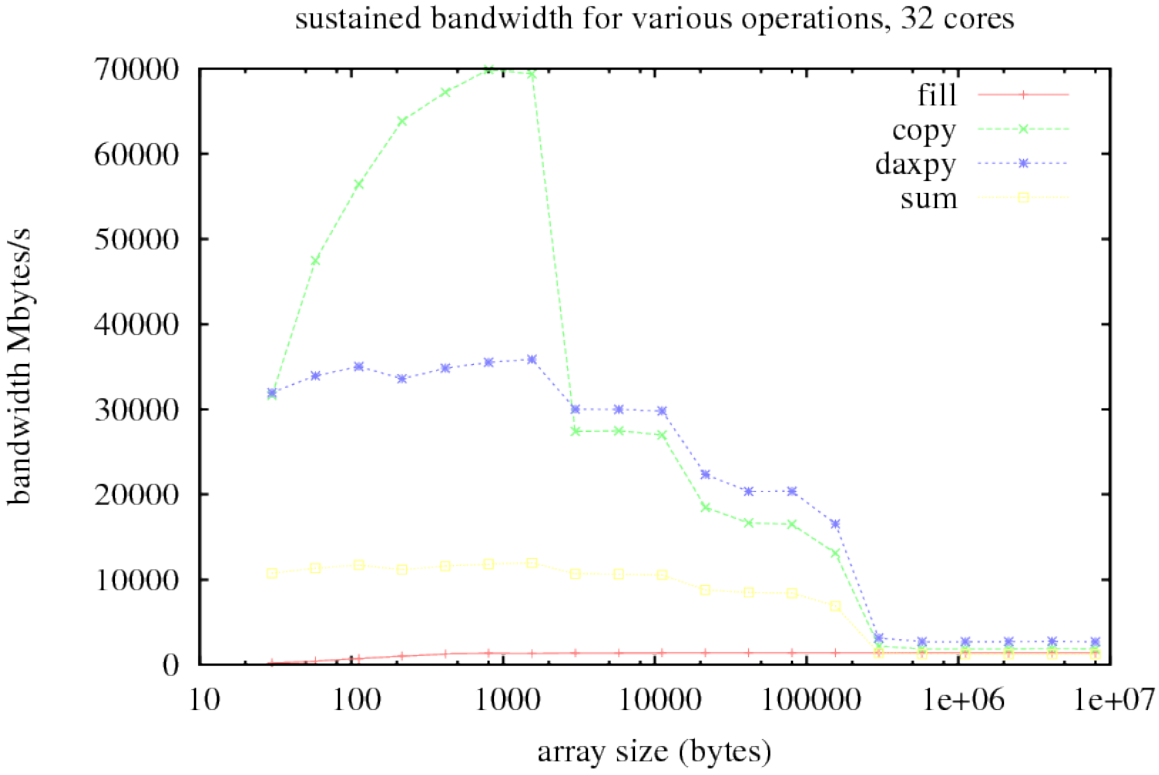

Figure 94: CURIE at TGCC - HPCC stream2 32 cores

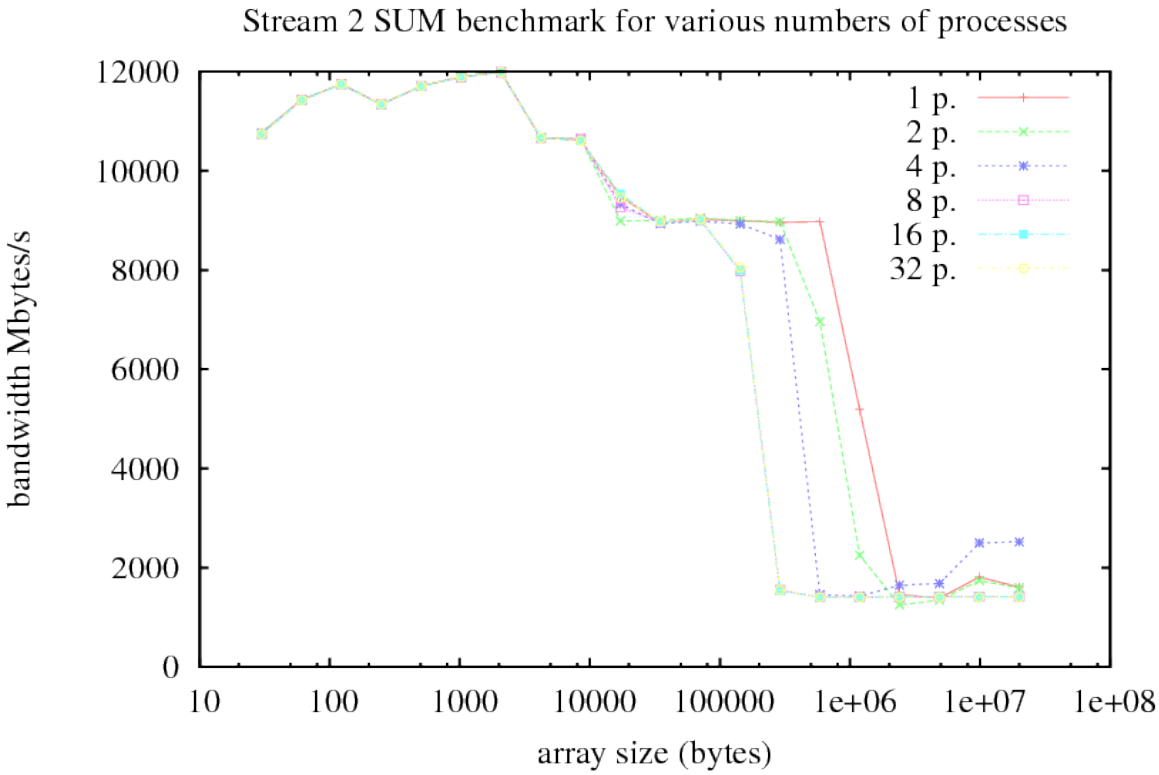

Figure 95: CURIE at TGCC - HPCC stream2 SUM

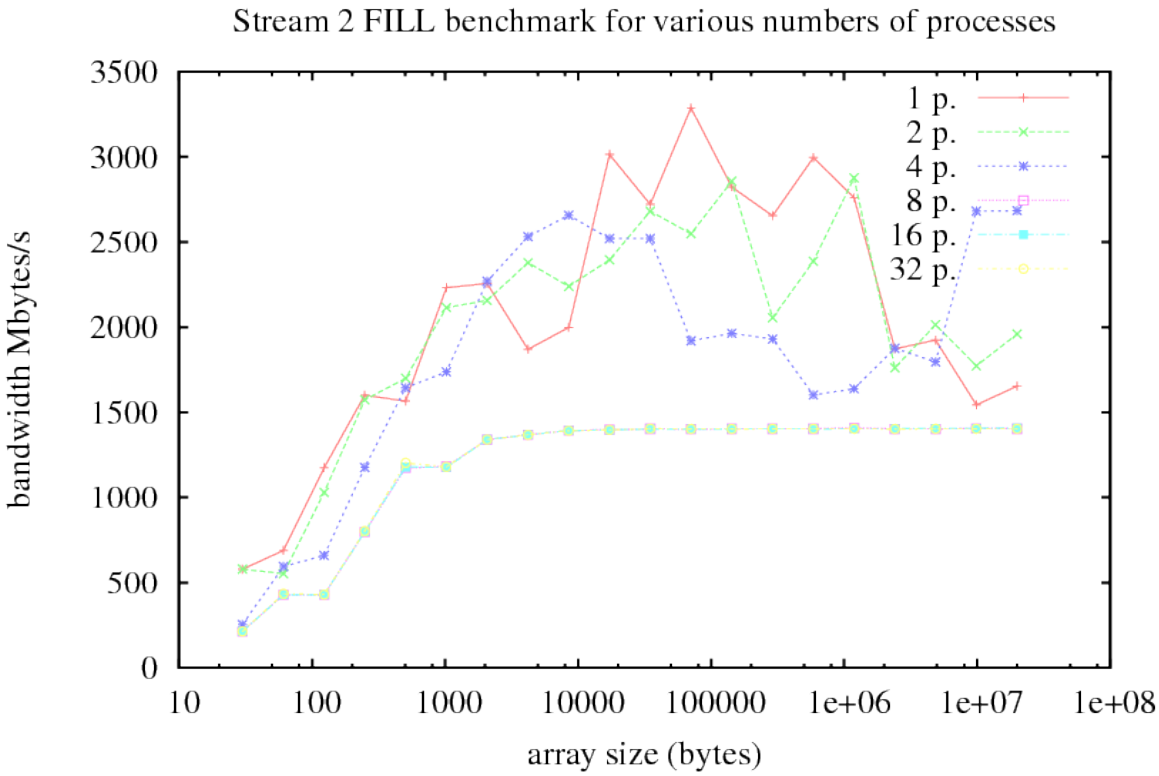

Figure 96: CURIE at TGCC - HPCC stream2 FILL

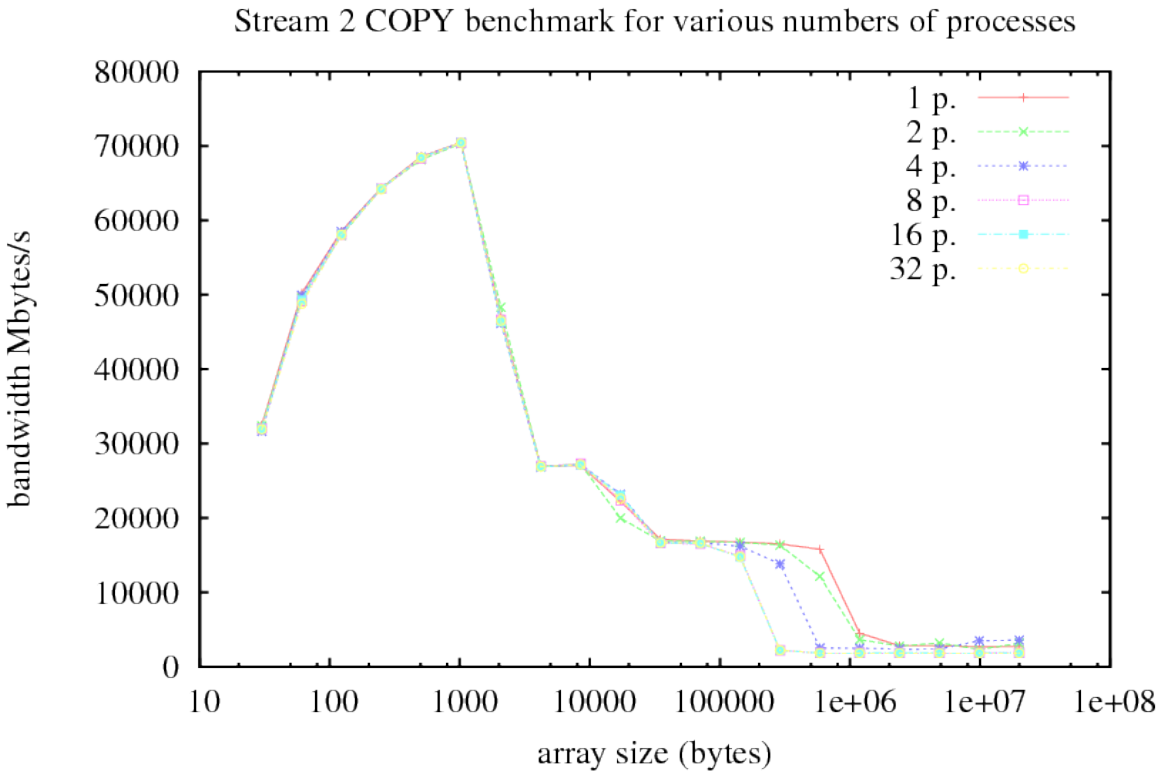

Figure 97: CURIE at TGCC - HPCC stream2 COPY

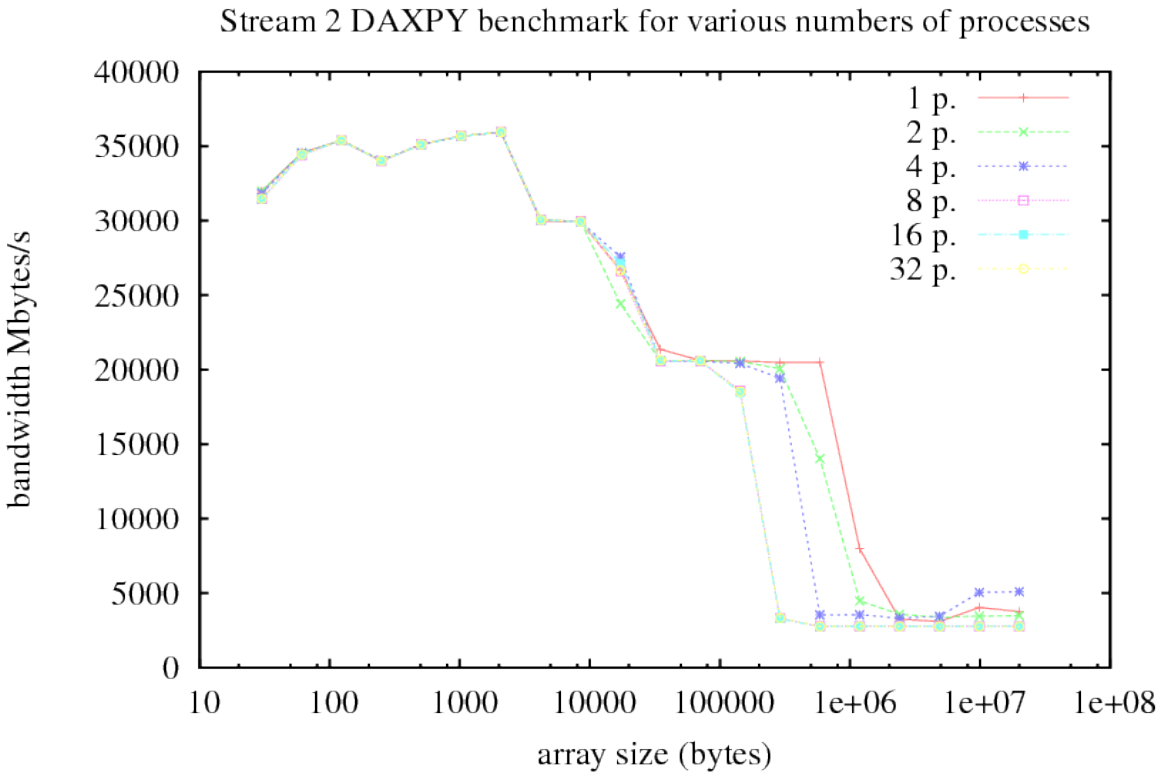

Figure 98: CURIE at TGCC - HPCC stream2 DAXPY

4.1.6 *T1.6 Cache Miss Performance*

The RandomAccess benchmark measures cache-miss performance by identifying the number of memory locations that can be randomly updated in one second. It reports one figure, Giga updates per second (GUP/s) and can be run on multiple cores and nodes. In this case, the tests were performed on a single node only.

General information

Compiler options:	-O3 -ftz -ip –ipo
Run on:	14,15,18,19 & 20-12-2011
Run by:	Soon-Heum „Jeff“ Ko

Results

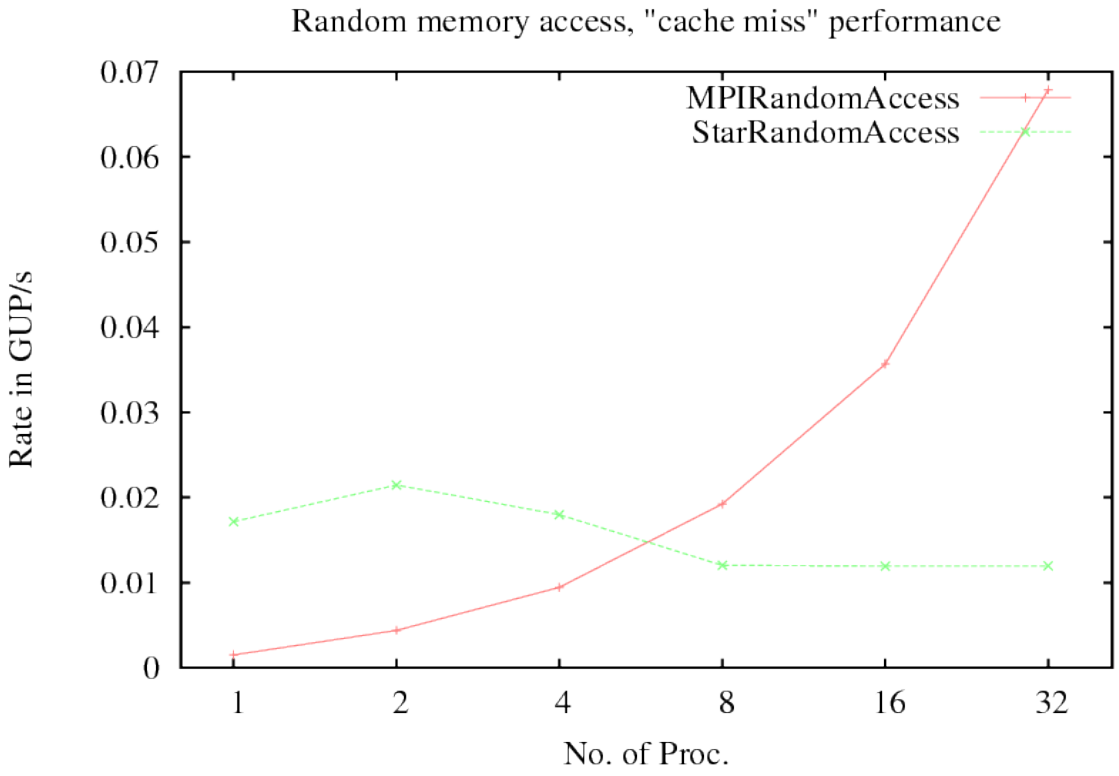

Figure 99: CURIE at TGCC - "cache miss" performance of StarRandomAccess and MPIRandomAccess

4.1.7 *T2.1 Memory Bandwidth Compared to Flop/s*

Memory access is commonly a restriction on computation speeds and this ratio quantifies the theoretical maximum bytes that can be delivered from memory for each flop. This is a derived ratio using previous results.

The run settings are provided where these results are first stated in this document, T1.1 for the LINPACK assessment and T1.4 for the STREAM memory bandwidth assessment, and so they are not repeated here.

Results

Cores/nodes	Processes	STREAM average bandwidth of 1 node (GB/s)	LINPACK (GFlop/s)	Byte/Flop
32/1	32	84.9728	247.400	0.34

4.1.8 *T2.2 MPI Bandwidth Compared to Flop/s at Different Scales*

This derived ratio provides an indicator of the number of network bytes which can be exchanged between processes for each flop. This ratio requires the all-to-all bandwidth measured with SKaMPI.

The SKaMPI results from the MPI all-to-all test have been used and compared to the LINPACK numbers (Gflop/s). Since there are no LINPACK results for 8 cores and for 2048 and 4096 cores, these values have been obtained by interpolation and extrapolation, respectively.

General information

Compiler options:	-O3 -ftz -ip -ipo
Run on:	10-12-2011 and 11-12-2011
Run by:	Carlos J. V. Simões (SKaMPI) Soon-Heum „Jeff“ Ko (LINPACK, see T1.1)

Results

Cores/nodes	SKAMPI Processes	all-to-all bandwidth (GB/s)	LINPACK (Gflop/s)	Byte/Flop
4/1	4	30.380	32.200	0.094347
8/1	8	51.376	64.024	0.080245
16/1	16	63.492	127.300	0.049876
32/1	32	99.749	247.400	0.040319
64/2	64	24.053	505.300	0.004760
128/4	128	31.561	1.018.000	0.003100
256/8	256	52.920	2.006.000	0.002638
512/16	512	101.451	3.998.000	0.002538
1024/32	1024	111.449	7.829.000	0.001424
2048/64	2048	207.964	15.330.976	0.001356
4096/128	4096	428.280	30.021.563	0.001427

Table 9 CURIE at TGCC - MPI bandwidth/s compared to Flop/s at different scales

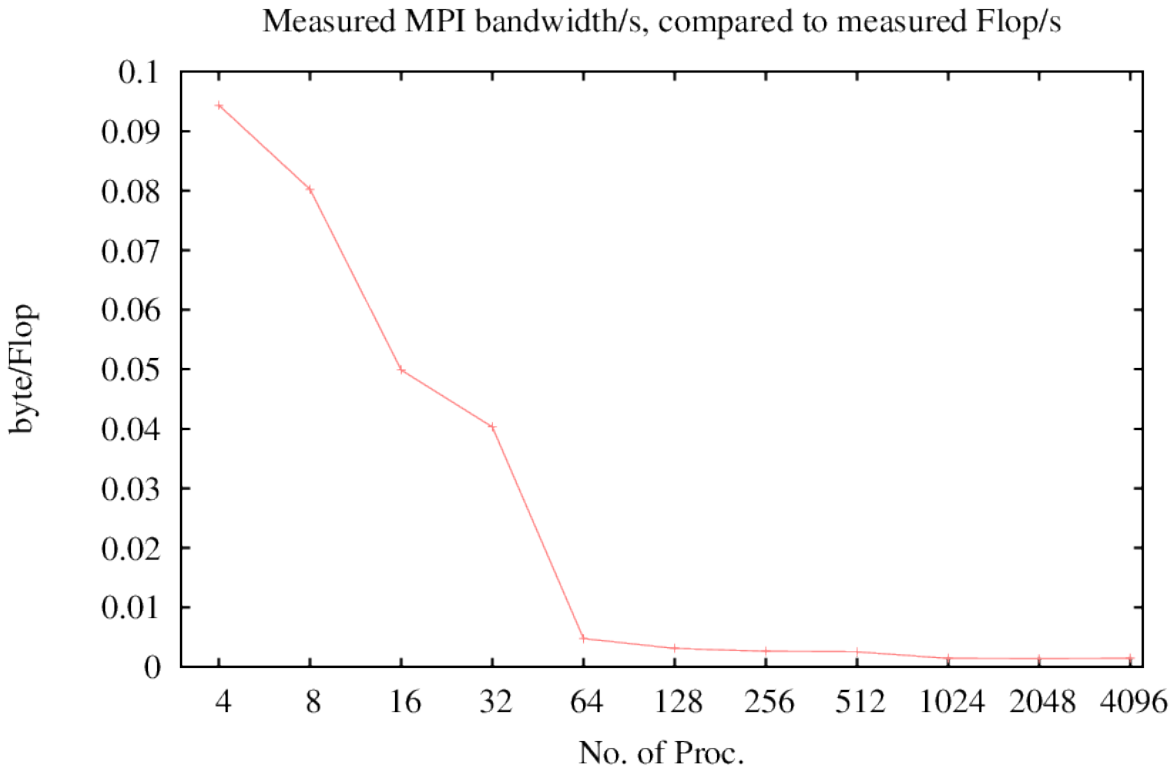

Figure 100: CURIE at TGCC - MPI bandwidth compared to Flop/s at different scales

4.1.9 *T2.3 Disk I/O compared to Flop/s at Different Scales*

This derived ratio provides a measure of the number of bytes that can be read and written for each flop for the global file system. Any fast local disk file system can also be reported. Where it is meaningful, typically where the compute node manages I/O, the ratio is reported at different scales. For this ratio the IOR benchmark was run to measure the I/O bandwidth.

For the disk performance test the fastest file system was chosen, /scratch (LUSTRE). A notable outlyer in the measurements is the single node, single core write speed, which is at least a factor 10 larger than the write speed for 2 cores on one node. At the time when the tests were run, the system was busy with jobs of other users. Since this is the shortest run of the set, lasting only 2 and a half minutes, it may have been rather quiet for a short period leading to this high speed.

General information

Compiler options:	-O3 -ftz -ip –ipo
Run on:	10-03-2012 and 11-03-2012
Run by:	Jeroen Engelberts (IOR) Soon-Heum „Jeff“ Ko (LINPACK, see T1.1)

Results

cores/ nodes	processes	Read Separated (GB/s)	Write Separated (GB/s)	LINPACK Gflop/s	Read Separated (Byte/Flop)	Write Separated (Byte/Flop)
1/1	1	0.03623	2.12979	8.215	0.004410	0.259256
2/1	2	0.03701	0.22021	16.264	0.002276	0.013540
4/1	4	0.06758	0.49131	32.200	0.002099	0.015258
8/1	8	0.13291	0.52734	64.024	0.002076	0.008237
16/1	16	0.24375	0.37607	127.300	0.001915	0.002954
32/1	32	0.53398	0.95293	247.400	0.002158	0.003852
64/2	64	0.86221	1.20713	505.300	0.001706	0.002389
128/4	128	1.73408	0.69902	1018.000	0.001703	0.000687
256/8	256	3.38389	1.66865	2006.000	0.001687	0.000832
512/16	512	3.42754	2.25498	3998.000	0.000857	0.000564

Table 10 CURIE at TGCC - Disk I/O compared to Flop/s at different scales

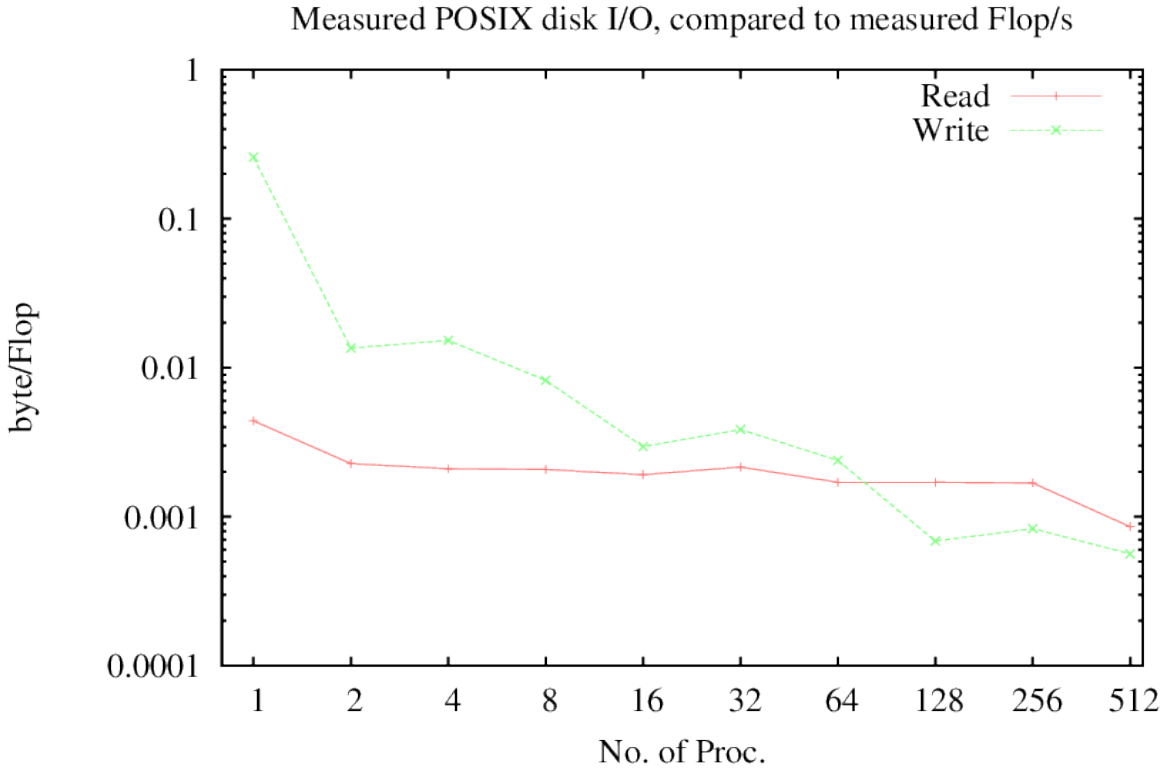

Figure 101: CURIE at TGCC - Disk I/O compared to Flop/s at different scales

4.1.10 *T3.1 Operating System Noise*

The operating system noise is measured with the P-SNAP benchmark [ref to P-SNAP] which runs a calibrated task and then reports the actual elapsed time of the task. The difference between the expected calibrated time and the actual time is a measure of operating system noise.

The idea is to run PSNAP on *all* nodes. This was not possible for the person performing the benchmark, because he is an ordinary user with no possibilities to select *all* nodes for the jobs.

Therefore, the histogram depicting noise per core over the *whole* system will not be present. Jobs with 1, 2, 4, 8 and 16 cores were run on node curie1257. The job with 32 cores was run on node curie1351.

General information

Compiler options:	-O3 –ftz -ip –ipo
Run on:	15-12-2011
Run by:	Soon-Heum „Jeff“ Ko

Results

Nodes	Tasks	Calibration Time (□s)	Mean Run Time (□s)	Standard Deviation of Run Time (□s)	Operating System Noise (%)
1	1,2,4,8,16 & 32	1000.00	1001.22	25.49	0.12

Figure 102 CURIE at TGCC - P-SNAP average operating system noise

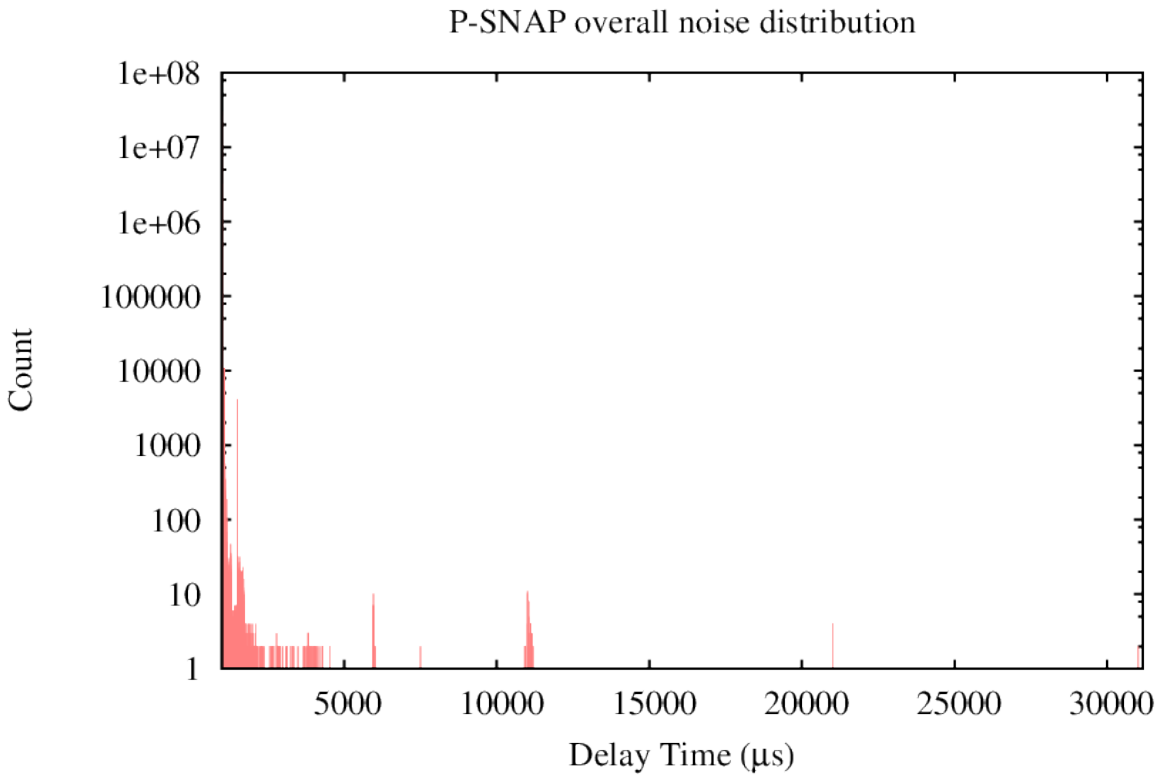

Figure 103: CURIE at TGCC - P-SNAP Operating System Noise (overall noise distribution)

4.1.11 *T3.2 Operating System Jitter*

The Selfish benchmark is designed as a noise recording tool which captures the delay profile for a particular core. Interruption events (called detours) are recorded with timestamp and duration.

General information

Compiler options:	-O3 –ftz –ip -ipo
Run on:	11-03-2012
Run by:	Jeroen Engelberts

Results

Cores/Nodes	Processes	Number of Events	Mean Event Time (□s)	Standard Deviation of Mean Event Time (□s)
32/1	32	1000	1938.4960	60.1741

Table 11CURIE at TGCC - Selfish detour events

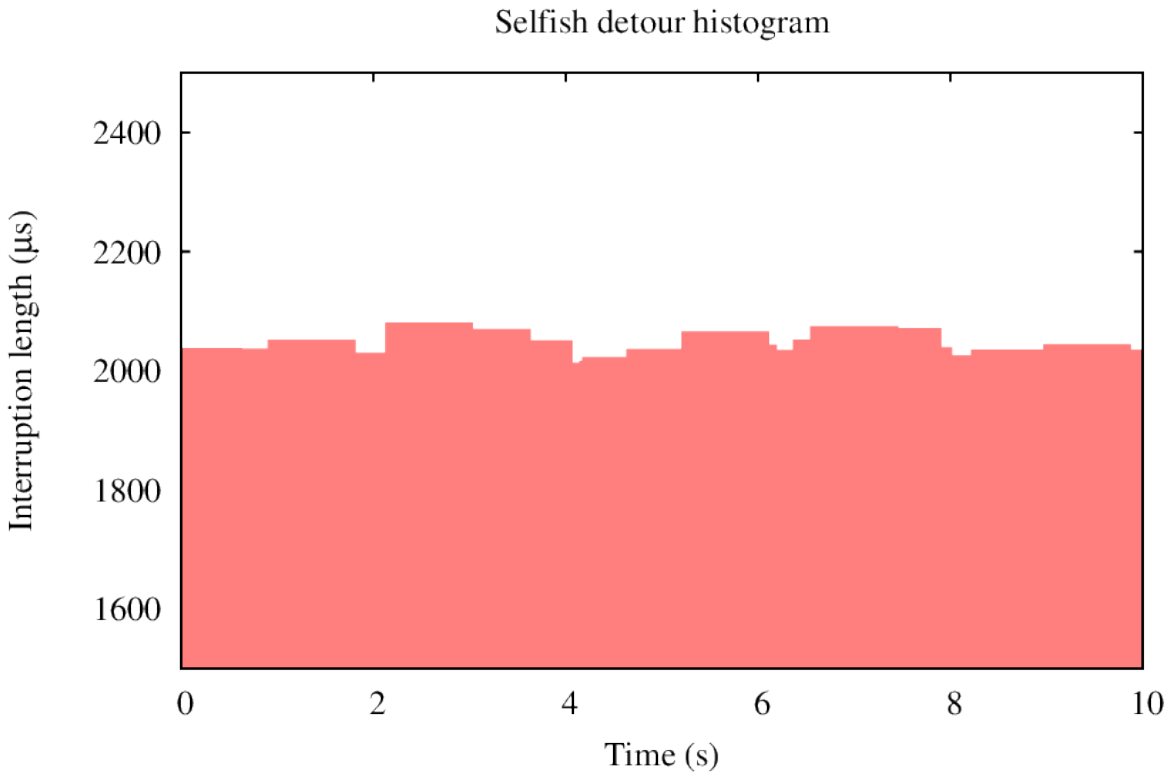

Figure 104: CURIE at TGCC - Selfish detour events

5 Performance Modelling

5.1 Motivation

The performance modelling of parallel and distributed systems has been, and continues to be, of great practical and theoretical importance in computer research in the design, development and optimization of computer and communication systems and applications. This is in part driven by the ongoing development of advanced computational architectures, which in turn is born out of a need for higher computational speed. The search for advanced computational architectures, sometime new, sometimes exotic, leads to architectural designs that accelerate the performance of applications into new realms, and require increased application or software complexity to take advantage of the new designs. These new architectures pose challenging problems that require new tools and methods to understand the impact on application performance in order to keep up with the rapid evolution and increasing complexity of such systems.

This process departs from the traditional approach of actually benchmarking available systems. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU, network bandwidth, and so on. Benchmarks provide a method of comparing the performance of various subsystems across different chip/system architectures and it is used to feed the performance models. Note that benchmarking is not usually used to estimate the performance of scientific codes, because it does not capture the complexity of the underlying algorithms used in the applications.

On the other hand, using performance models, systems that are not currently installed and hence not available for measurement/benchmarking can be analysed and their achievable performance can be quantified. This is the biggest advantage of performance modelling. The performance of applications running on these future systems can be analyzed, predicted with high accuracy, and optimized.

Moreover, in general, performance models can be used throughout the life-cycle of systems from first design through to maintenance. Models can be used during the procurement of new systems to compare the performance from different system proposals. Models have also been used to analyze systems during installation in order to identify performance issues and assist in system optimizations. A performance model may analyse both application and system characteristics, depending on the accuracy of the model. Application characteristics are defined uniquely for each application and include processor flow, data structures used, frequency of use and mapping onto the system, and their potential for resource contention. System characteristics include node configuration (processors per node, shared resources), and inter-processor communication (latency, bandwidth, topology).

In particular, in this study we focused on identifying performance issues of scientific applications and assessing the sensitivity of applications to system parameters. The latter study of the codes will quantify the impact on the code of variations in performance of the underlying hardware in the systems that might be available in the forthcoming future.

In addition, there is a growing interest in the scientific community to scale up scientific applications to a large number of processes. For this reason it is necessary to perform a performance analysis of selected codes in order to understand the details of the performance phenomena involved. This performance analysis not only allows us to study the current behaviour of the codes, but also emphasises bottlenecks, in order to identify the most promising areas for performance improvement at large scale.

Problems such as excessive synchronization, communication overhead, and algorithmic inefficiencies are common in all types of causes of performance scaling issues. Algorithmic inefficiencies might involve a wide range of issues such as load imbalance – an uneven distribution of work across all compute nodes – or a centralized communication pattern prone to generate network contention.

Performance analysis tools are usually used to analyze the characteristic of underlying algorithms in applications. Examples of popular tools are Paraver[1], Kojak[8], SCALASCA[9], Vampir[10], and TAU[11]. These tools rely on performance profilers to collect data, including hardware interrupts, code instrumentation, operating system calls, performance hardware counters, and communication operations. Nearly all parallel performance analysis tools include at least some rudimentary method for analysing algorithmic inefficiencies, providing a quick view of what is going on in the application on the system where the application was running.

On the other hand, in order to analyse the performance on systems not currently available performance models of the applications or simulation tools can be used. In the first approach, performance models of the computation can be developed using, for example, a curve fitting approach. This technique models the times that the application spends both in computation and communication, using the data collected via the performance profilers. Applications are run on different numbers of processors so we can collect enough information about the computation and communication costs and predict, based on this data, what would happen at larger scales. A common example of this method is Prophesy[12].

In the second approach, using simulation tools such as Dimemas[13] more accurate predictions can be obtained because the communications of the applications can be modelled in very high detail. As in the previous approach, these tools take data collected via profilers and simulate the behaviour of the application from the beginning to the end of the execution. This can provide a more accurate view of the performance during the application execution. spotting particular performance issues in some particular region of the code that might be hidden by the curve fitting approach. These simulation tools provide a configurable parallel platform which users can customized to their needs in order to analyse the performance in systems that are not available today. In particular, the interconnect can be parameterised by bandwidth and latency, and the computation can be scaled out to model faster CPUs.

In this study, a detailed performance characterization was conducted on two important scientific codes, Quantum Espresso and CP2K. These codes are popular and widely used in the Material Science domain. They are relevant for the scientific community and for PRACE because their scientific challenges that they can potentially tackle. Both codes are the subject of optimisation efforts in other tasks of WP7.

5.2 Performance profiling and simulation tools

In order to collect performance data from actual runs from applications we used the tracing tool Extrae[14] and the visualization and analysis tool Paraver. The combined use of Extrae and Paraver offers an enormous analysis potential, both qualitative and quantitative. With these tools the actual performance bottlenecks of parallel applications can be identified. The microscopic view of the program behavior that the tools provide is very useful to optimize the parallel program performance.

Extrae is a dynamic instrumentation package to trace programs compiled and run with the shared memory model (e.g. OpenMP or pthreads), the message passing (MPI) programming model or both programming models (different MPI processes using OpenMP or pthreads within each MPI process). Detailed information is gathered from applications such as

information of MPI calls, processors and network hardware counters, and application callers which are the routine addresses present in the process stack at any given moment during the application run. Callers can be used to link the trace file with the source code of the application. Extrae generates trace files that can be later simulated with Dimemas and visualized with Paraver as well.

Paraver is a flexible parallel program visualization and analysis tool based on an easy-to-use Motif GUI. Paraver was developed responding to the need of having a qualitative global perception of the application behavior by visual inspection and then to be able to focus on the detailed quantitative analysis of the problems. Paraver provides a large amount of information useful to decide the points on which to invest the programming effort to optimize an application. Expressive power, flexibility and the capability of efficiently handling large traces are key features addressed in the design of Paraver. The clear and modular structure of Paraver plays a significant role in achieving these targets. Some Paraver features are the support for detailed quantitative analysis of program performance, concurrent comparative analysis of several traces, fast analysis of very large traces, and support for mixed message passing and shared memory (network of SMPs).

Performance predictions of applications in future systemd have been evaluated using Dimemas, a performance analysis tool for message-passing programs. The Dimemas simulator reconstructs the time behavior of a parallel application on a machine modeled by a set of performance parameters such as available network bandwidth and network latency. Thus, performance experiments can be done easily. The supported target architecture classes include networks of workstations, single and clustered SMPs, distributed memory parallel computers, and even heterogeneous systems. For communication, a linear performance model is used, but some non-linear effects such as network conflicts are taken into account. The simulator allows specifying different task to node mappings, as well as various processing speeds for the computation bursts of applications. Dimemas generates trace files that are suitable for the performance analysis tools Paraver.

Moreover, we also provide a model of the computation and the communication of the applications. The computation model is obtained through the curve fitting approach based on the existing scaling data. On the other hand, the communication models are parameterised using the standard communication model LogGP for the dominant communication operations in the application. The LogGP model consists of the following parameters:

- **L** is an upper bound on the Latency of a send operation from one processor to another.
- **o** is the overhead, i.e., the time that the host processor is engaged in the transmission or reception of a message.
- **g** is the gap between two consecutive messages. It defines the minimum time-interval between two message sends or receptions.
- **G** is the Gap per byte for long messages. It defines the time needed to transmit a single byte for the bulk- transfer of long messages.
- **P** is the number of involved Processors.

Applications are run and profiled on Marenostrum hosted by Barcelona Supercomputer Center, Spain. Marenostrum consists of 2,560 JS21 compute nodes (blades) and 42 p615 servers. Every blade has two IBM Power PC 970MP processors (2.3 GHz), running Linux operating system with 8 GB of memory RAM and 36 GB local disk storage. All the servers provide a total of 280 TB of disk storage accessible from every blade through GPFS (Global Parallel File System). The total number of cores in the system is 10,240. JS21 blades are

interconnected with both Myrinet (used by applications) and Gigabit Ethernet interconnects (used by the file system).

5.3 Performance Modelling: Quantum Espresso

Quantum Espresso is an integrated suite of computer codes for electronic-structure calculations and materials modelling at the nano-scale. It is based on density-functional theory, plane waves, and pseudo-potentials (both norm-conserving and ultrasoft). The version 4.3.1 released on April, 2011 was used in the experiments.

The pw.x code of Quantum Espresso was used in the evaluation which is the basic code for the self-consistent cycle (SCF) for structure optimization molecular dynamic simulations. We have compiled the application for the POWERPC architecture of Marenostrum using the flag **`ppc64`**. We used the compiler IBM XL C/C++ for Linux, v10.1 with optimization flag **`-O3`** and MPI version MPICH-MX v1.2.7.

The input deck used in the evaluation is the standard input ausurf112 that consists of a surface of 112 gold atoms. This is a typical strong scaling problem where the problem size per processor is proportionally reduced as the number of processors is increased.

We have profiled a section of the code in order to limit the size of the trace files. This section of the code corresponds to where Quantum Espresso spends most of the time, which is the **`electrons`** routine. This routine is the driver of the self-consistent cycle, which in turn uses the routine **`c_bands`** for computing the bands at fixed Hamiltonian, the routine **`sum_band`** to compute the charge density, the routine **`v_of_rho`** to compute the new potential and the routine **`mix_rho`** to mix input and output charge densities. The profiled data corresponds to one iteration of this routine.

We have profiled Quantum Espresso on various numbers of MPI processes, ranging from 32 up to 512. We have allocated four MPI processes per node using the full node in the Marenostrum supercomputer.

5.3.1 *Scalability Analysis*

This section analyses the scalability of this application based on the profiled data and builds models of the computation and communication to predict the performance at larger scales.

The computation model is obtained via curve fitting the existing data, and communication models are obtained using a standard cost model for the collective functions.

Figure 105 shows the runtime of Quantum Espresso for different numbers of processors on the Marenostrum supercomputer. We plotted the measured runtime and also the ideal runtime which is calculated by simply dividing up the runtime obtained at 32 processes by the ratio of the number of processors and 32. This ideal runtime represents a lower bound when scaling the code, and thus it will tell us how far we are from the optimal runtime.

As we can see, the measured runtime is significantly reduced as we increase the number of processors. However, we can see that there is a divergence in between the measured runtime and the ideal runtime and this difference is growing with the number of processors. In particular, the difference in between the ideal and the measured runtimes is 11%, 16%, 25%, and 67% at 64, 128, 256, and 512 processors, respectively. This is usually attributed to the cost of the communications and load imbalance in the code. This growing difference is an indication of the poor scalability of the code.

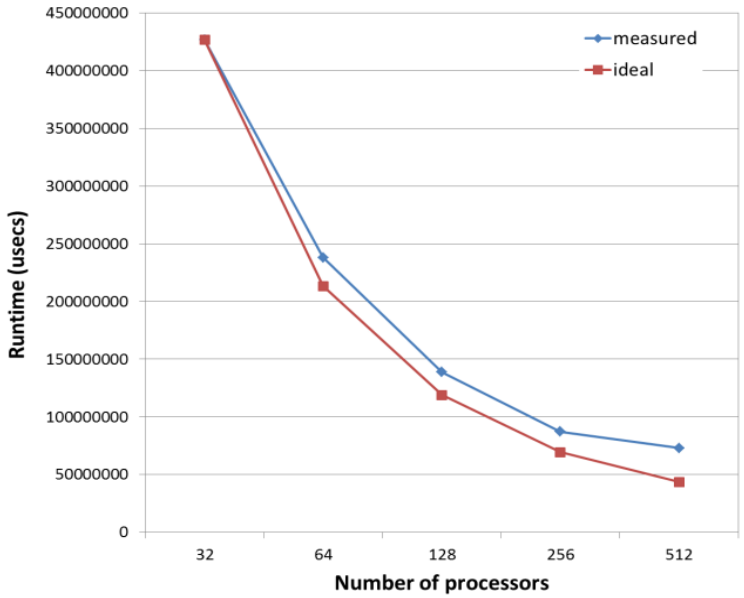

Figure 105: Runtime of Quantum Espresso on Marenostrum.

Figure 106 shows the resulting parallel efficiency of Quantum Espresso for various numbers of processors. The parallel efficiency is a measure to describe the fraction of the time in an application that is being used by the processors doing computation. The ideal parallel efficiency should be 100% regardless of the number of processors. As we can see, the parallel efficiency significantly drops as we increase the number of processors. At 512 processors, the parallel efficiency is very low: only 20%. This is clearly showing the scalability problems of this application which can be due to either load imbalance and/or high communication times.

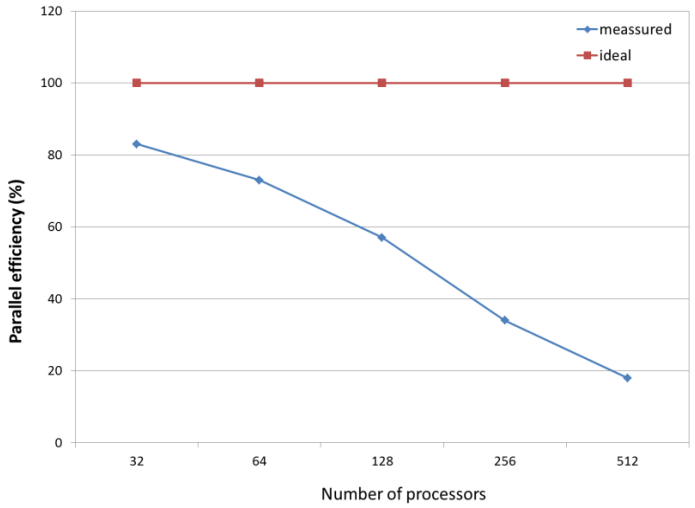

Figure 106: Parallel efficiency of Quantum Espresso

5.3.2 *Modelling Computation time*

Figure 107 shows the maximum accumulated computation time measured across all the MPI processes for various numbers of processors, and the associated ideal computational time. The computation time is the time of the code without taking into account any communication related time. We take the maximum computation time across all the MPI processes because it represents the slowest process in the application. As before, the ideal time is calculated by simply dividing up the computational time obtained at 32 processes by the ratio of the number of processors and 32. As can be seen, the computation time is significantly reduced as we increase the number of processors, but still is not equal to the expected ideal computation time. This behaviour has been also observed before with the runtime. However, here we can see that the difference in between the measured and the ideal times is smaller than before because we do not have the effect of the communications. Specifically, there is a difference of 2%, 10%, 34%, and 38% at 64, 128, 256, and 512 processors. This mismatch is typically solely attributed to the load imbalance of the code.

Using curve fitting we can model the computation time of the code. This is also shown in Figure 107. The computation time (in usecs) has been approximated by using a polynomial function of third order:

$$T_{computation} = -4E+06x^3 + 6E+07x^2 - 3E+08x + 7E+08$$

where $x=\log_2 p-4$, and p is the number of processors.

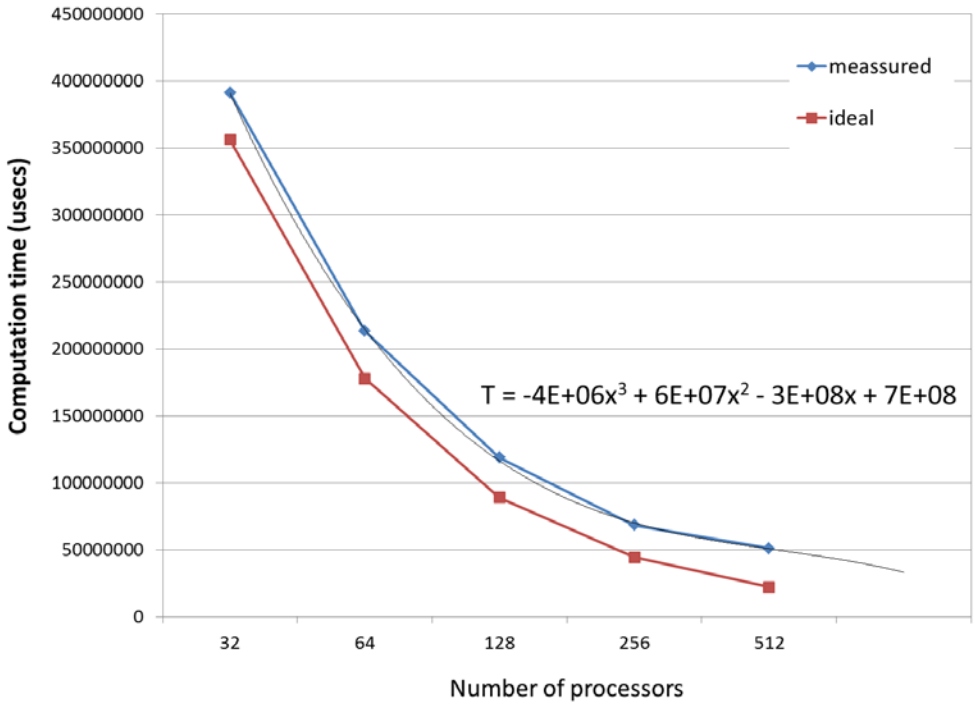

Figure 107: Maximum computation time measured in Marenostrum and its associated expected ideal computation time.

5.3.3 *Load Balance Analysis*

Figure 108 shows the load balance ratio for various numbers of processors. The load balance ratio represents the ratio between the average total computation time across all MPI processes and the maximum total computation time. This is an estimate of how well the computation is

balanced across processes: a low value means an unbalanced load. As we can see, the load imbalance grows significantly as the number of processes increases. In particular, we obtained a load balance of 0.91, 0.82, 0.67, 0.43, and 0.27 at 32, 64, 128, 256, and 512 processes. This is the main factor that prevents this code from achieving a perfect scaling of the computation, as noted above.

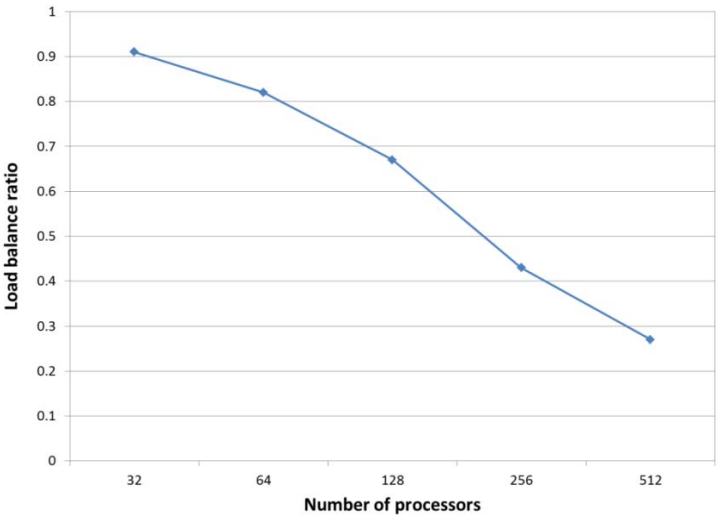

Figure 108: Load balance ratio for various numbers of processors.

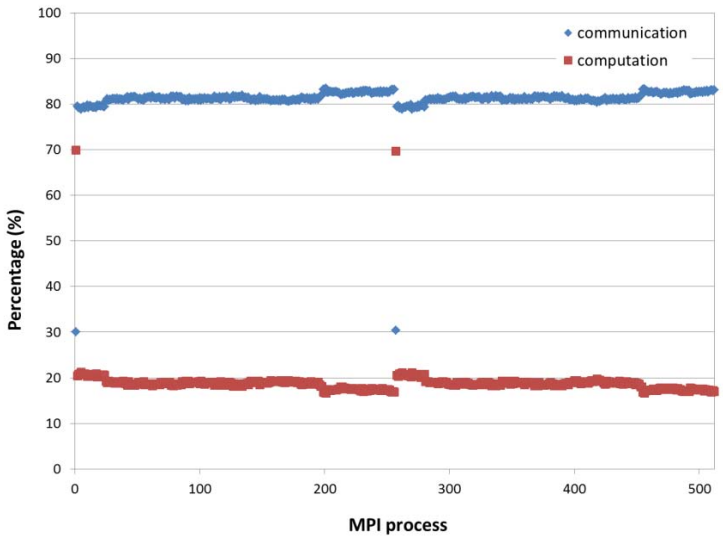

Figure 109: Percentage of time that each MPI process spends on communication and computation

Figure 109 shows more detailed information of the computation balance for each MPI process on the 512-process run. It is shown the percentage of time that each MPI process spends on computation and communication. As you can see, there are two processes (1 and 257) where the computation takes four times longer. Moreover, there are some groups of processes that are doing around 15% less computation than the rest. The groups are from 197 to 256 and from 455 to 512. This load imbalance is preventing the application from scaling out.

5.3.4 *Modelling Communication*

In this section, we analyse the communication operations of the code and develop a simple model to capture the behaviour at the scale.

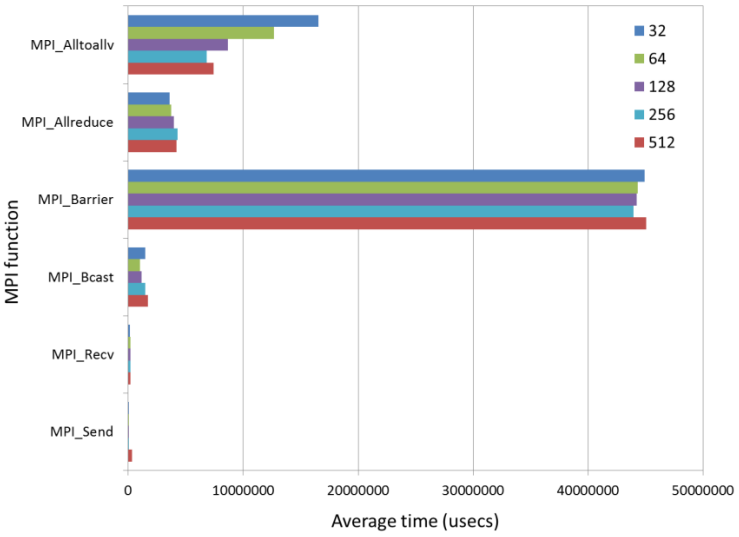

Figure 110: Average time spent in each MPI call for various processor counts.

Figure 110 shows the accumulated average time spent per process on each MPI call. The code spends most of the time in MPI_Barrier and MPI_Alltoallv, which account for 67% and 25% of the total communication time respectively. Also, notice that the time spent on the MPI_AlltoAllv call is gradually reduced as the number of processors increases. The reason for this is that the data of this collective operation is reduced as the number of processors increases. This is illustrated in Figure 111 which shows the size of the data for this collective operation on various numbers of processes. As we can see, on 32 processes the size of the data is around 300KB which reduces in proportion to the number of processes. The data size can be estimated using the curve fitting approach as follows,

$$S_{MPI\_AlltoAllv} = 608256e^{-0.693x}$$

where $x=\log_2 p-4$, and p is the number of processors.

The MPI_Barrier operation clearly dominates the communication time. The reason for this may be due to the load imbalance of the code where some processes are waiting much longer on the barrier to others to finish their computation.

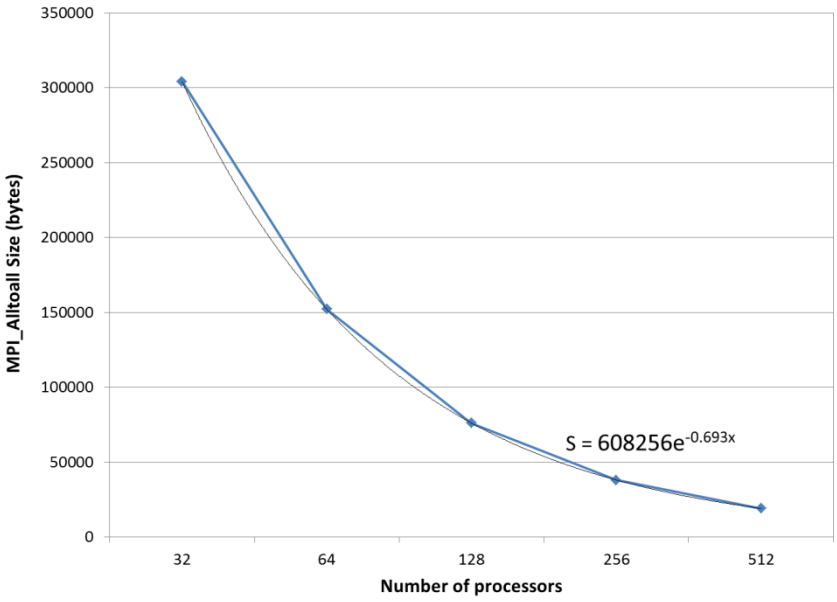

Figure 111: Size of the MPI_AlltoAll for various processor counts.

Figure 112 shows the number of calls performed per each MPI function. As can be seen, the MPI_Barrier and MPI_AlltoAllv functions are the ones called most, accounting for 4,100 and 3,700 calls, respectively. MPI_Barrier operations are collective communication functions used to synchronize processes which do not involve the transfer of any data among processes. It guarantees that by the end of the operation, the remaining processes have at least entered the barrier. It is suggested to the application developers to change their algorithms in order to reduce the need for such synchronization.

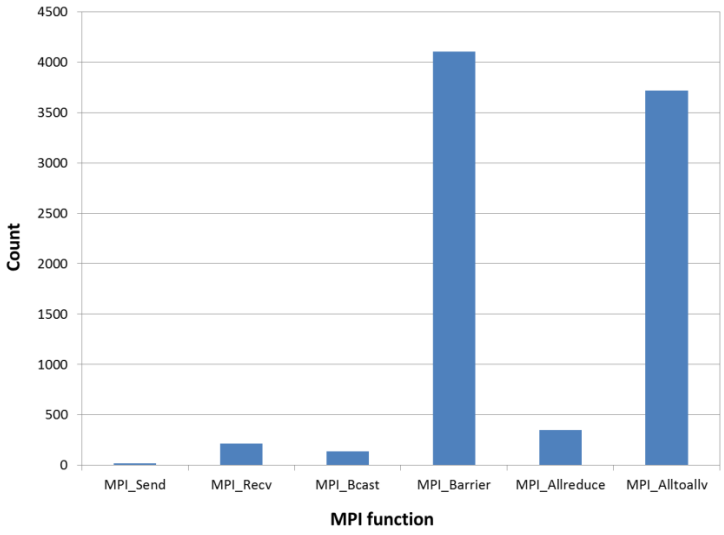

Figure 112: Average number of calls performed per each MPI function

The communication model for this code is composed by the aggregation of the corresponding communication models for the MPI_Barrier and MPI_AlltoAllv communication functions. The typical implementation of an MPI_Barrier follows the Bruck algorithm that requires $\log_2 P$ steps, where P is the number of processes. In this algorithm, at a step k, process r receives a

zero byte message from, and sends messages to processes $(r - 2^k)$ and $(r + 2^k)$ (with wrap around), respectively. Therefore, the total time of this communication would be,

$$T_{MPI\_Barrier} = \log_2(p) \times (L + o + g) \times 4100$$

For the case of the MPI_AlltoAllv the typical algorithm is used is a pairwise exchange which requires P-1 steps to complete. Therefore, the total time of this communication function would be,

$$T_{MPI\_AlltoAllv} = (p - 1) \times (L + o + (m - 1) \times G + g) \times 3700$$

where *m* is the size of data exchanged in the collective which has been estimated before. Notice, that at large scale the MPI_AlltoAllv is going to dominate the communication time in a perfectly load balanced application. On the other hand, the MPI_Barrier will dominate because of the load imbalance of the application which is the major bottleneck to achieve a higher scalability.

5.3.5 *System parametric studies*

In this section, we are going to explore the performance of this application on future machines where the processing speed might be higher and also the network might provide higher a bandwidth. This study is conducted by simulation of the application trace on the Dimemas simulator. Various CPU increase ratios with respect to the CPU speed of the Marenostrum supercomputer are assumed following an exponential growth in powers of two. Also various network bandwidth are considered from 1Gbps/s up to 64Gbps following an exponential growth in powers of two as well.

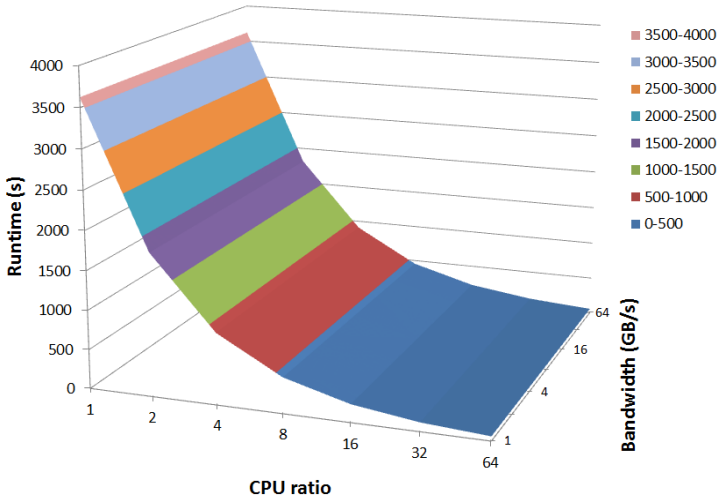

Figure 113: Sensitivity to CPU speed and network bandwidth

Figure 113 shows the resulting runtime of Quantum Espresso when varying the CPU speed and network bandwidth on a 512-process run. Network latency is assumed to be zero in the simulations. As can be seen, the application will be benefit most from the increase of the CPU

ratio and not from the increase of the network bandwidth. The increase in performance is proportional to the increase of the CPU ratio. The reason for this is that this application suffers from load imbalance, and thus an increase of the CPU speed is directly correlated with a reduction in the runtime.

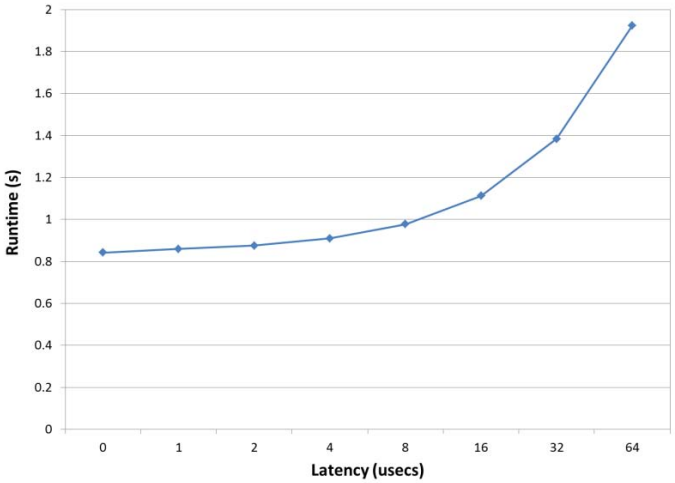

Figure 114: Sensitivity to network latency

Figure 114 shows the runtime of Quantum Espresso for different network latency ranging from 0 usecs up to 64 usecs for the case of the 512-process run, together with an increase by a factor of 64 in both the CPU speed and the network bandwidth. As can be seen, the network latency significantly impacts the performance of Quantum Espresso, an increase of a factor of 64 times in the latency degrades the performance by 124%. The reason for this is that this application has collective operations which are sensitive to latency.

5.4 Performance Modelling: CP2K

CP2K is a freely available (GPL) program, written in Fortran 95, to perform atomistic and molecular simulations of solid state, liquid, molecular and biological systems. It provides a general framework for different methods such as e.g. density functional theory (DFT) using a mixed Gaussian and plane waves approach (GPW), and classical pair and many-body potentials. The cp2k-2_2-branch version released on 23 October 2011 was used in these experiments.

The executable in this code is called cp2k.popt. We have compiled the application for the POWERPC architecture of Marenostrum using the flag **`ppc64`**. We used the compiler IBM XL C/C++ for Linux, v10.1 with optimization flag **`-O3,`** using MPI version MPICH-MX v1.2.7, and with the following libraries: LAPACK v3.2.1 and BLAS v1. The input deck used in the evaluation is the standard input H2O-32 that consists of 32 water molecules. This is a typical strong scaling problem where the problem size per processor is proportionally reduced as the number of processors is increased.

We have profiled a section of the code in order to limit the size of the trace files. This section of the code corresponds to where CP2K spends most of the time. This is the Quickstep SCF located in the function **`scf_env_do_scf`**. The profiled data corresponds to one iteration of this routine.

We have profiled CP2K on various numbers of processors, ranging from 32 up to 512 MPI processes. We have allocated four MPI processes per node, using the full node in the Marenostrum supercomputer.

5.4.1 *Scalability Analysis*

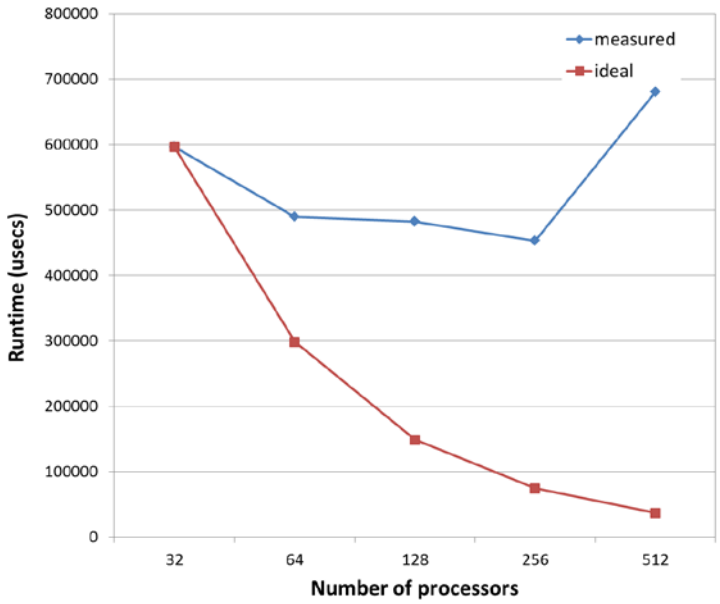

Figure 115: Runtime of CP2K measured in Marenostrum and its associated expected ideal runtime.

Figure 115 shows the measured runtime and the expected ideal runtime of CP2K when running in 32 up to 512 processes. As you can see, the measured runtime decreases as we increase the number of processors from 64 up to 256 processors. The improvement on the

execution time is very modest only 6% when running on 256 with respect to 128 processes. This shows the poor scaling properties of this application. Furthermore, at 512 processes we no longer obtain a reduction on the execution time: there is actually an increase of 50% of the execution time from 256 to 512 processes. We will investigate in more detail of the cause of this in the following sections.

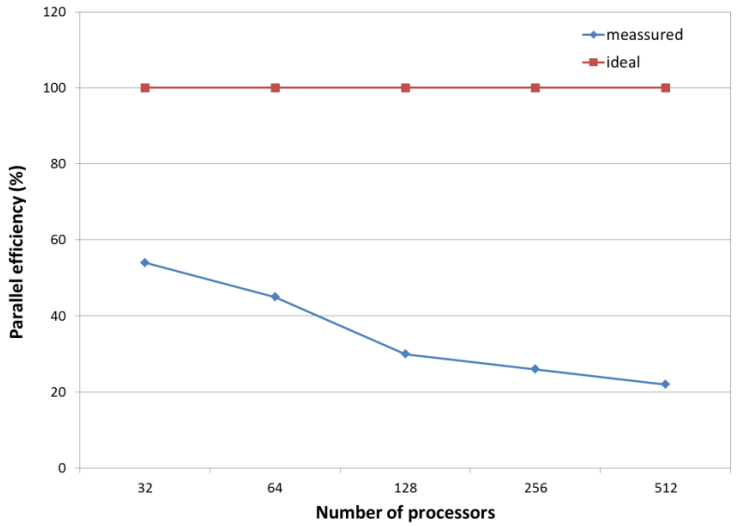

Figure 116: Parallel efficiency of CP2K

Figure 116 shows the resulting parallel efficiency of CP2K for various numbers of processors. The ideal parallel efficiency should be 100% regardless of the number of processors. The parallel efficiency is a measure to describe the fraction of the time in an application that is being used by the processors doing computation. As you can see, the parallel efficiency is very low even on very small number of processes. In particular, at 32 processes the parallel efficiency is only 54%, and drops to 22% at 512 processes.

5.4.2 *Modelling Computation Time*

Figure 117 shows the maximum accumulated computation time among all processes for various numbers of processors. Also, it is showed the ideal computation time which assumes that the computation decreases proportionally with the number of processes. As can be seen, the computation time is no further reduced at 256 and 512 processes. This lack of scalability in the computation time seems to be a major cause that prevents to scale this code to larger number of processes.

The computation time can be modelled using curve fitting which is shown in the Figure 117. The computation time (in usecs) has been approximated by using a polynomial function of second order:

$$T_{computation} = 18959x^2 - 148184x + 472594$$

where $x=\log_2 p-4$, and p is the number of processors.

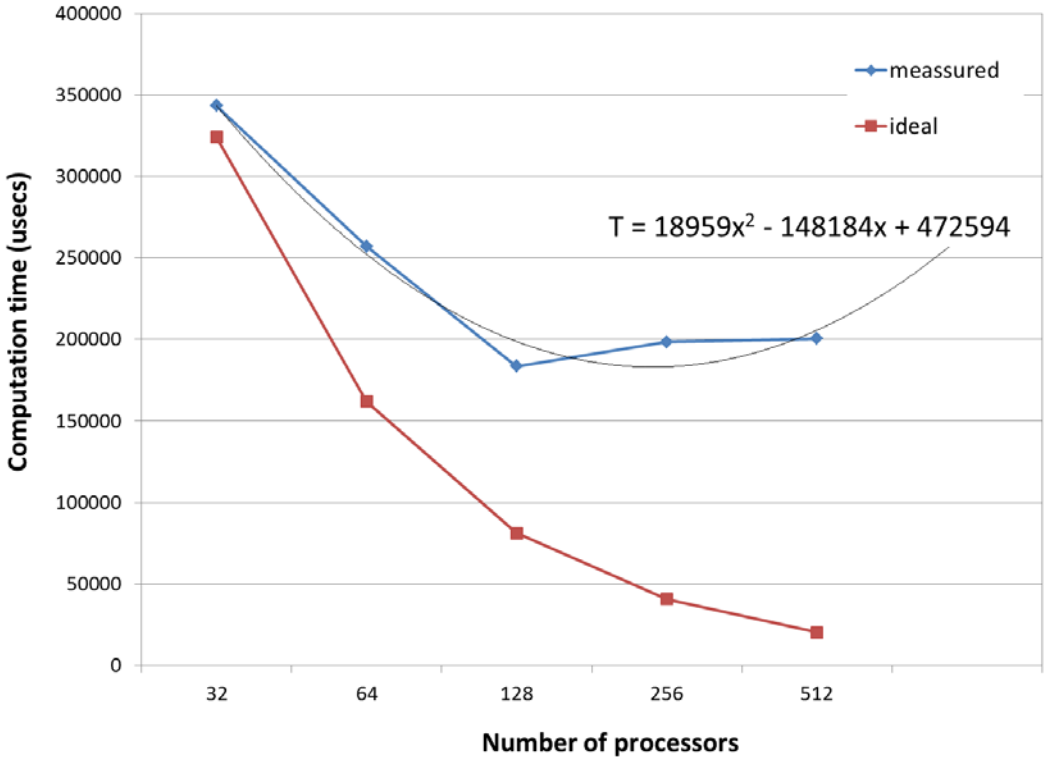

Figure 117: Maximum computation time measured in Marenostrum and its associated expected ideal computation time.

5.4.3 *Load Balance Analysis*

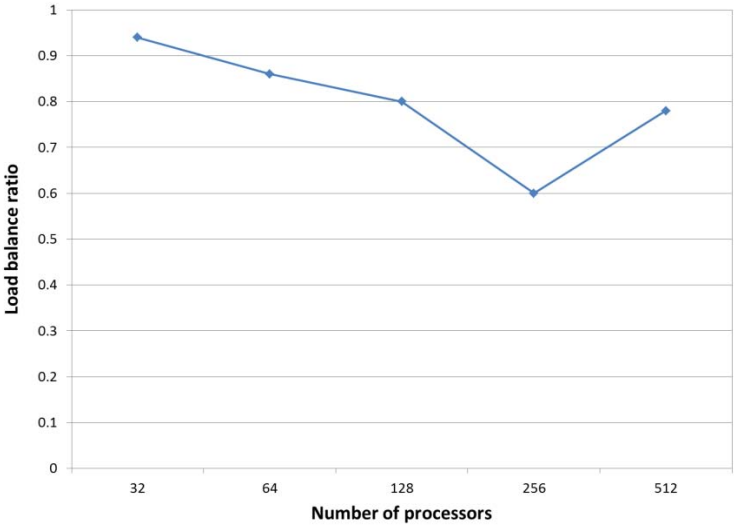

Figure 118: Load balance ratio for various numbers of processors.

Figure 118 shows the load balance ratio for the same number of processes. The load balance ratio represents the ratio in between the average total computation time across all MPI processes and the maximum total computation time (shown in Figure 117). This is an estimation of how well the computation is balanced across processes: a low value means an unbalanced load. The load imbalance grows significantly as the number of processes

increases. In particular, we obtained very low load balance ratios of 0.6, and 0.78 at 256, and 512 processes respectively, showing again the scalability problems of this code.

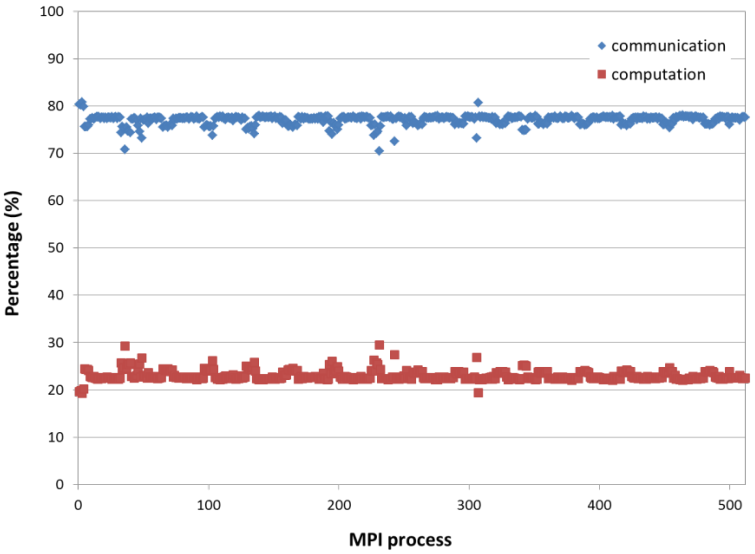

Figure 119: Percentage of time that each MPI process spends on communication and computation

Figure 119 shows more detailed information of the computation balance for each MPI process on the 512 process run. It is shown the percentage of time that each MPI process spends on computation and communication. As you can see, the computation is quite unbalanced: we can find some groups of processes that are performing much more computation that the rest of them. In particular, the number of processes in each group is always eight processes and they are spaced 24 processes apart from each other group. They perform between 10-28% more computation the other processes. Moreover, there are some processes that are performing much less computation that the rest of the processes. Processes from 1 to 4 and process 307 are performing around 15% less computation. These differences in computation are clearly causing the load imbalance observed in the previous section.

5.4.4 *Modelling Communication*

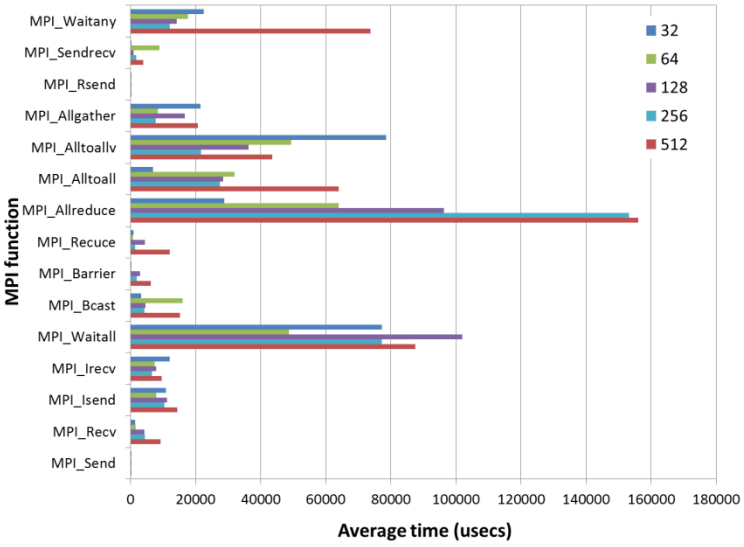

Figure 120: Average time spent in each MPI call for various processor counts

Figure 120 shows the accumulated average time spent per process in each MPI call. The code spends most of the time in MPI_Allreduce. This is a collective operation which is well known to cause scalability problems to many scientific applications. Furthermore, it is observed that time spent in this function increases with the number of processes which may become the one of the major bottlenecks to scale out this code to larger number of processes, even if the load imbalance problem is fixed.

Also, notice that the point-to-point functions MPI_Waitany and MPI_Waitall are the second most predominant communication functions at all process counts. However, there is no clear trend that suggests that these functions will become the bottleneck at large process counts. In addition, Figure 121 shows the maximum message size for the point-to-point communication functions for various numbers of processors. As can be seen, the message size gradually decreases with the number of processors: at 512 processes the message size is only 256 bytes.

Particular emphasis should be given to the MPI_Waitany function at 512 processors. We can see that we substantially increase the time spend in this function with respect to 256 processes. This appears to be an anomaly that happens in the system during the execution of this application due, most probably to a bad communication channel. A high error count on this link could generate a substantial increase in the time to receive some messages. This in likely to be the cause of the higher execution time previously observed at 512 processors in

Figure 115.

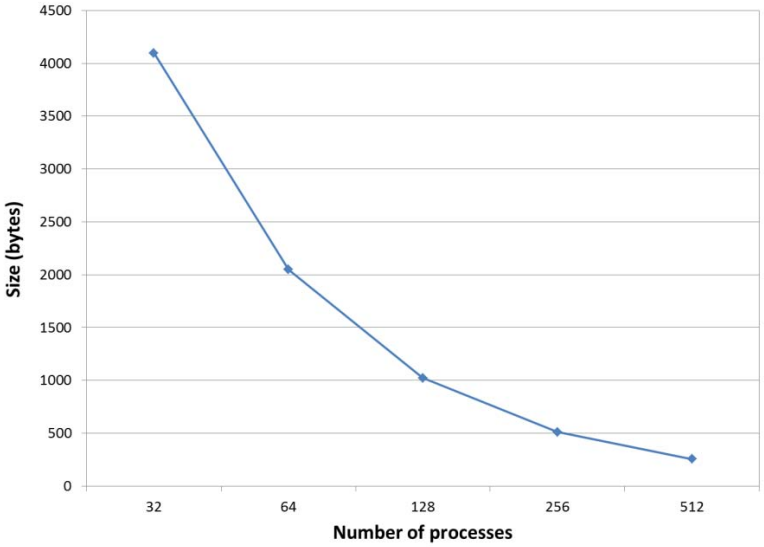

Figure 121: Maximum message size for the MPI point-to-point functions

Figure 122 shows the number of times that the communication functions are called at 512 processes. It is seen, that the largest number of calls are coming from MPI_Waitall, accounting for 4,500 calls. However, we have seen that the code is not spending much time on this call. On the other hand, MPI_Allreduce, the most critical function for scaling is only called 69 times.

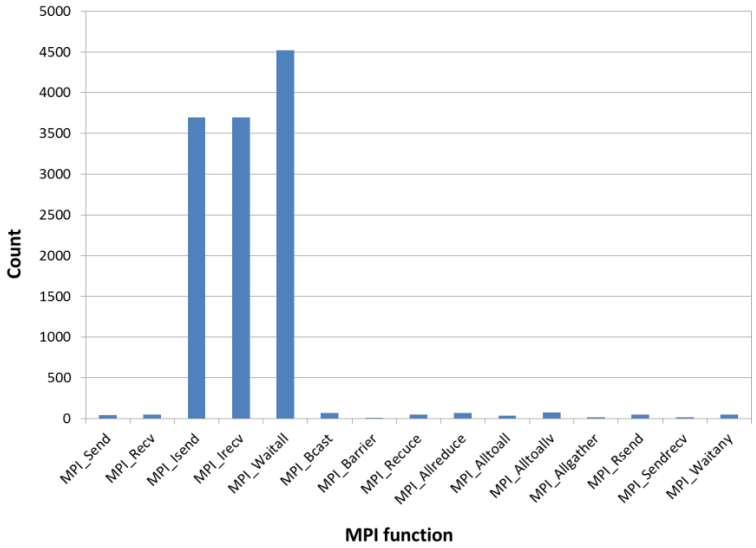

Figure 122: Average number of calls performed per each MPI function

The typical implementation of an MPI_Allreduce follows the binomial algorithm that requires $\log_2 P$ steps, where P is the number of processes. Therefore, the total time of this communication for this code would be

$$T_{MPI\_Allreduce} = \log_2(p) \times (L + o + g) \times 69$$

5.4.5 *System parametric studies*

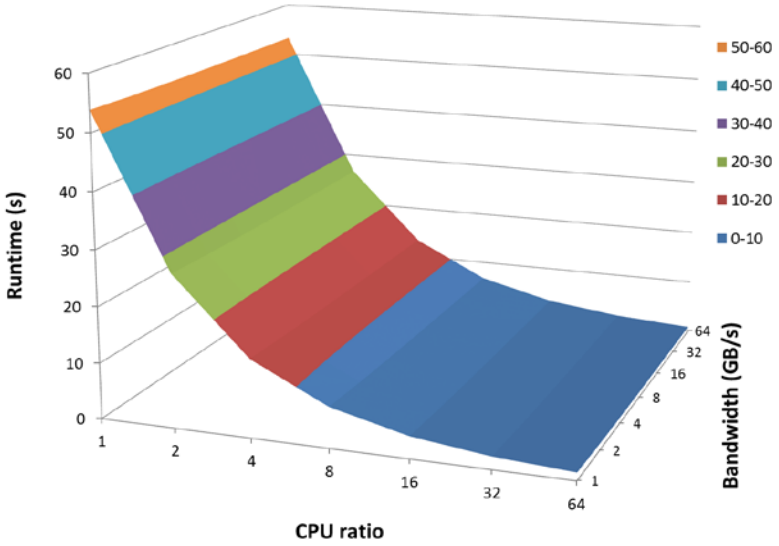

Figure 123: Sensitivity to CPU speed and network bandwidth

Figure 123 shows the resulting runtime of CP2K when varying the CPU speed and network bandwidth on a 512-process run. Network latency is assumed to be zero in the simulations. As can be seen, the application will be benefit most from the increase on the CPU ratio and not

from an increase in network bandwidth. The increase in performance is proportional to the increase of the CPU ratio. The reason for this is that this application suffers from load imbalance, and thus an increase of the CPU speed is directly correlated with a reduction in the runtime.

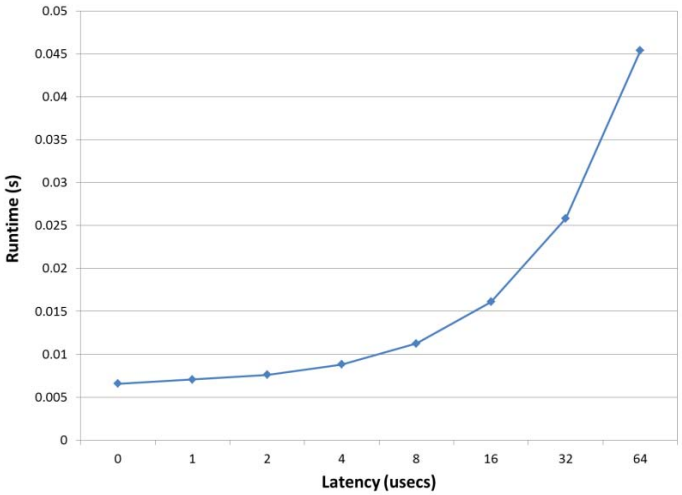

Figure 124: Sensitivity to network latency

Figure 124 shows the runtime of CP2K for different network latencies ranging from 0 usecs up to 64 usecs, for the case of the 512-process run together with an increase by a factor of 64 in both the CPU speed and the network bandwidth. As can be seen, the performance CP2K is very sensitivity to the network latency. For example, an increase of network latency by a factor of 64 degrades the performance by a factor of 5.5. This is an expected result since the MPI_Allreduce collective communication operation is the dominant communication operation which is very sensitive to network latency.

6 Conclusions and Future Work

6.1 Benchmarking

We have updated the PRACE Application Benchmark Suite developed in the Preparatory Project and performed the runs of PABS and the synthetic benchmarks on the Tier-0 systems. We have shown significant progress: nearly all codes could be run on the new Tier-0 system CURIE and also most of the codes have been ported to the IBM BlueGene/P system, some of them for the first time. It turned out that we had to skip one code, because the code was too special for general benchmarking, but on the other hand we could show an increased scaling for other codes. In general, we found that codes achieve a higher percentage of peak performance on the Bull x86 Cluster than on the IBM BlueGene/P for small partition sizes, but that scalability tends to be better on the IBM BlueGene/P. This demonstrates that application benchmark results can show significant differences between systems, and give useful guidance for users in choosing which system to run their codes on.

All these results are valuable outcomes which can be used in the follow up tasks of PRACE-2IP. In this project a publically available Unified European Application Benchmark Suite (UEABS) will be created and the results of PRACE-1IP are essential for this work: there are up-to-date benchmark results available on the current Tier-0 systems, we gained further experience on porting issues of the codes, and on the status of licensing. Additionally the work performed was valuable for the development of the codes under consideration because lessons learnt can lead to improvements in the codes.

We have also run a set of synthetic benchmarks on the Bull x86 Cluster system, which provide additional information, and add one further system to the data collected in the Preparatory Phase Project.

For future work it will still be necessary to perform updates on the application codes and input sets to keep the results of the benchmark suite significant. This work is currently in progress in Task 7.4 of the PRACE-2IP. In the meantime, the PABS suite can be downloaded from the PRACE-SVN server for internal use. The general use of the benchmark suite has been made possible through the integration in the benchmark environment JuBE: this will continue in UEABS. UEABS results also have the potential to be used to convert CPU hours costs between different PRACE systems in the future.

6.2 Performance Modelling

Performance models and simulation studies on two of the PABS applications, Quantum Espresso and CP2K, have been conducted in order to accurately predict the behaviour of applications in future machines. These applications have been run in order to collect key performance data using commonly used performance tools that are used to identify scalability problems and to characterize its behaviour in future systems.

We have found in Quantum Espresso that there are two major scalability problems: one is high load imbalance and the other is excessive synchronization. For the case of CP2K, this application shows also high load imbalance, though less than in Quantum Espresso, and heavy use of collective communication operations. Both applications would benefit from higher CPU speeds and lower network latency, but not from having higher network bandwidth.

There is a strong interest in PRACE to assess the requirements of applications, in particular the ones previously listed in Section 2.2 which have been already ported in PRACE-1IP WP7.4. We already did the analysis to two of them Quantum Espresso and CP2K. Therefore,

for future work, it would be interesting to extend this analysis to more applications making also special emphasis at the scalability issues of these applications.

7 Annex A

In this Annex the raw data of the benchmark runs can be found.

7.1 CODE_SATURNE

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	1,74	111,48	0,005153
1024	3,48	71,53	0,004015
2048	6,96	45,45	0,003160

Table 12: Results for Code_Saturne, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
128	1,19	158,75	0,005292
256	2,38	91,67	0,010909
512	4,76	61,63	0,003408
1024	9,52	56,63	0,001854

Table 13: Results for Code_Saturne, Test Case A on Bull x86 Cluster

7.2 CP2K

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	1,74	552,42	0,001040
1024	3,48	375,36	0,000765
2048	6,96	327,03	0,000439

Table 14: Results for CP2K, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
32	0,30	1037,1	0,003240
64	0,60	613,1	0,002741
128	1,19	410,2	0,002048
256	2,38	287,3	0,001462
512	4,76	227,6	0,000923
1024	9,52	298,1	0,000352

Table 15: Results for CP2K, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	0,87	1496,23	0,000768
512	1,74	903,68	0,000636
1024	3,48	587,27	0,000489
2048	6,96	477,47	0,000301

Table 16: Results for CP2K, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
64	0,60	1202,5	0,001397
128	1,19	760,3	0,001105
256	2,38	516,1	0,000814
512	4,76	387,0	0,000543
1024	9,52	372,8	0,000282

Table 17: Results for CP2K, Test Case B on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	127,7	0,002249
2048	6,96	97,5	0,001473
4096	13,93	82,7	0,000868
16384	55,71	75,5	0,000238

Table 18: Results for CP2K, Test Case C on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
128	1,19	149,4	0,005623
256	2,38	87,0	0,004828
512	4,76	56,8	0,003697
1024	9,52	51,7	0,002031

Table 19: Results for CP2K, Test Case C on Bull x86 Cluster

7.3 CPMD

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	4,76	69,87	0,003006
1024	9,52	44,19	0,002376
2048	19,95	23,31	0,002252

Table 20: Results for CPMD, Test Case A on Bull x86 Cluster

7.4 EUTERPE

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
128	0,44	53,06	0,043306
256	0,87	26,97	0,042599
512	1,74	13,90	0,041327
1024	3,48	7,37	0,038972
2048	6,96	4,10	0,035027
4096	13,93	2,46	0,029189

Table 21: Results for EUTERPE, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
128	1,19	8,49	0,098946
256	2,38	4,43	0,094814
512	4,76	2,55	0,082358

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	9,52	1,56	0,067312
2048	19,95	1,06	0,049531
4352	40,47	0,89	0,029496

Table 22: Results for EUTERPE, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	1,74	103,41	0,005555
768	2,61	69,30	0,005526
1280	4,35	41,73	0,005506
1536	5,22	34,80	0,005502
2816	9,57	19,37	0,005392
5632	19,15	9,85	0,005302
11520	39,17	5,01	0,005096
11776	40,04	4,91	0,005087
23296	79,17	2,69	0,004695
46848	159,28	1,53	0,004103
47059	160,00	1,51	0,004139

Table 23: Results for EUTERPE, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	32,39	0,012968
512	4,76	16,31	0,012876
1024	9,52	8,43	0,012456
2048	19,95	4,57	0,011489
4352	40,47	2,39	0,010338
8704	80,95	1,63	0,007579
8960	83,33	1,59	0,007548

Table 24: Results for EUTERPE, Test Case B on Bull x86 Cluster

7.5 GADGET

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	7,99	0,035948
2048	6,96	5,84	0,024591
4096	13,93	6,83	0,010513
8192	27,85	16,63	0,002159

Table 25: Results for GADGET, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	9,52	6,92	0,015174
2048	19,95	6,42	0,001814
4096	38,09	8,63	0,003042

Table 26: Results for GADGET, Test Case B on Bull x86 Cluster

7.6 GROMACS

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
128	0,44	1333,65	0,001723
256	0,87	695,98	0,001651
512	1,74	374,27	0,001535
1024	3,48	211,41	0,001359
2048	6,96	124,26	0,001156
4096	13,93	80,67	0,000890
8192	27,85	65,68	0,000547

Table 27: Results for GROMACS, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
32	0,30	781,20	0,004302
64	0,60	403,40	0,004169
128	1,19	198,17	0,004243
256	2,38	111,55	0,003750
512	4,76	64,84	0,003231
1024	9,52	36,78	0,002838
2048	19,95	23,81	0,002188
4096	38,09	29,14	0,000905

Table 28: Results for GROMACS, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	0,87	610,27	0,001883
512	1,74	334,28	0,001720
1024	3,48	196,83	0,001458
2048	6,96	133,54	0,001072
4096	13,93	99,89	0,000718
8192	27,85	88,86	0,000403

Table 29: Results for GROMACS, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
32	0,30	567,78	0,005916
64	0,60	293,81	0,005715
128	1,19	160,22	0,005250
256	2,38	89,54	0,004667
512	4,76	188,96	0,001111
1024	9,52	38,42	0,002763
2048	19,95	24,81	0,002100
4096	38,09	26,44	0,000101

Table 30: Results for GROMACS, Test Case B on Bull x86 Cluster

7.7 NAMD

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
16	0,05	415,56	0,044235
32	0,11	223,56	0,041113
64	0,22	117,76	0,039025
128	0,44	63,14	0,036392
256	0,87	36,06	0,031861
512	1,74	24,82	0,023145
1024	3,48	29,44	0,009756

Table 31: Results for NAMD, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
16	0,15	117,84	0,05703
32	0,30	78,18	0,04298
64	0,60	39,21	0,04285
128	1,19	23,87	0,03520
256	2,38	13,66	0,03075
512	4,76	12,85	0,01634

Table 32: Results for NAMD, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
32	0,11	1276,38	0,00720
64	0,22	667,87	0,00688
128	0,44	357,79	0,00642
256	0,87	187,70	0,00612
512	1,74	105,43	0,00545
1024	3,48	68,06	0,00422
2048	6,96	52,82	0,00272
4096	13,93	53,53	0,00134

Table 33: Results for NAMD, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
32	0,30	321,95	0,01044
64	0,60	171,36	0,00981
128	1,19	100,61	0,00835
512	4,76	82,05	0,00256

Table 34: Results for NAMD, Test Case B on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
32	0,30	1643,84	0,002044
64	0,60	911,99	0,001842
128	1,19	477,94	0,001758
256	2,38	255,76	0,001642
512	4,76	185,53	0,001132

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	9,52	94,22	0,001114
2048	19,05	92,98	0,000565

Table 35: Results for NAMD, Test Case C on Bull x86 Cluster

7.8 NEMO

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	4,76	4,08	0,05147
1024	9,52	2,30	0,04566
2048	19,95	1,80	0,02917
3072	28,57	1,72	0,02035
4096	38,09	1,29	0,02035

Table 36: Results for NEMO, Test Case A on Bull x86 Cluster

7.9 NS3D

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
320	1,09	22,8	0,04040
1280	4,35	11,6	0,01986
2560	8,70	9,3	0,01238

Table 37: Results for NS3D, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
320	2,98	6,3	0,05334
1280	11,90	1,2	0,06830
2560	23,81	0,6	0,06667

Table 38: Results for NS3D, Test Case A on Bull x86 Cluster

7.10 QCD

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	317,00	0,000906
2048	6,96	141,40	0,001016
4096	13,93	75,75	0,000948
8192	27,85	43,21	0,000831
16384	55,71	22,74	0,000789
32678	111,41	16,33	0,000550
65536	222,82	12,32	0,000364

Table 39: Results for QCD, Kernel A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	389,80	0,001078
512	4,76	205,80	0,001020
1024	9,52	110,10	0,000954
2048	19,95	71,25	0,000737
4096	38,09	63,66	0,000412

Table 40: Results for QCD, Kernel A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	129,40	0,002220
2048	6,96	64,93	0,002212
4096	13,93	32,71	0,002195
8192	27,85	16,66	0,002155
16384	55,71	8,69	0,002066
32678	111,41	4,45	0,002017
65536	222,82	2,46	0,001824

Table 41: Results for QCD, Kernel B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	70,31	0,005974
512	4,76	35,06	0,005990
1024	9,52	17,04	0,006162
2048	19,95	8,85	0,005933
4096	38,09	6,56	0,004002
8192	76,19	4,97	0,002641

Table 42: Results for QCD, Kernel B on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	115,90	0,008628
2048	6,96	114,70	0,008718
4096	13,93	113,80	0,008787
8192	27,85	115,50	0,008658
16384	55,71	115,00	0,008696
32678	111,41	113,30	0,008826
65536	222,82	117,50	0,008511

Table 43: Results for QCD, Kernel C on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	52,75	0,018957
512	4,76	48,69	0,020538
1024	9,52	55,88	0,017895
2048	19,95	58,50	0,017094

Table 44: Results for QCD, Kernel C on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	1,48	0,194070
2048	6,96	0,75	0,191228
4096	13,93	0,42	0,170967
8192	27,85	0,22	0,163196
16384	55,71	0,15	0,119677

Table 45: Results for QCD, Kernel D on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	1,34	0,313453
512	4,76	0,79	0,265840
1024	9,52	0,43	0,244202
2048	19,95	0,23	0,228275
4096	38,09	0,14	0,187512
8192	76,19	0,11	0,125008

Table 46: Results for QCD, Kernel D on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	3,48	280,00	0,001026
2048	6,96	142,70	0,001006
4096	13,93	71,93	0,000998
8192	27,85	39,32	0,000913
16384	55,71	21,25	0,000845
32678	111,41	13,04	0,000688
65536	222,82	7,04	0,000637

Table 47: Results for QCD, Kernel E on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	332,00	0,001265
512	4,76	180,00	0,001167
1024	9,52	88,74	0,001183
2048	19,95	46,47	0,001130
4096	38,09	23,69	0,001108

Table 48: Results for QCD, Kernel E on Bull x86 Cluster

7.11 QUANTUM_ESPRESSO

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
4096	13,93	143,9	0,000499
8192	27,85	87,0	0,000413
16384	55,71	59,4	0,000302
24576	83,56	50,0	0,000239
32678	111,41	48,7	0,000184
49152	167,12	55,9	0,000107

Table 49: Results for Quantum_Esoresso, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	4,76	688,0	0,000305
1024	9,52	328,5	0,000320
2048	19,95	176,0	0,000298
4096	38,09	128,0	0,000205
8192	76,19	195,0	0,000067

Table 50: Results for Quantum_Espresso, Test Case A on Bull x86 Cluster

7.12 WRF

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
735	2,50	6,26	0,063923
1470	5,00	3,37	0,059371
2941	10,00	1,74	0,057475
5882	20,00	0,95	0,052635

Table 51: Results for WRF, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	3,31	0,126896
512	4,76	1,67	0,125757
1024	9,52	0,94	0,111709
2048	19,95	0,53	0,099063
4096	38,09	11,96	0,002195

Table 52: Results for WRF, Test Case A on Bull x86 Cluster

7.13 ALYA

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
735	2,5	98,84	0,004049
1471	5,0	53,89	0,003710
2941	10,0	29,47	0,003393
5882	20,0	17,82	0,002806
11765	40,0	11,56	0,002163
23529	80,0	7,02	0,001781
47059	160,0	3,89	0,001607

Table 53: Results for ALYA, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
274	2,55	499,77	0,000785
549	5,11	254,32	0,000770
1097	10,20	139,27	0,000704
2194	20,40	79,93	0,000613
4388	40,81	62,10	0,000395

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
8777	81,63	49,94	0,000245

Table 54: Results for ALYA, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
735	2,5	15,20	0,026326
1471	5,0	8,65	0,023115
2941	10,0	3,36	0,029764
5882	20,0	2,59	0,019306
11765	40,0	1,20	0,020833
23529	80,0	0,89	0,014045
47059	160,0	0,53	0,011792

Table 55: Results for ALYA, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
274	2,55	92,81	0,004228
549	5,11	53,83	0,003638
1097	10,20	36,84	0,002661
2194	20,40	31,10	0,001576
4388	40,81	33,68	0,000728
8777	81,63	30,55	0,000401

Table 56: Results for ALYA, Test Case B on Bull x86 Cluster

7.14 AVBP

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	4,76	367,58	0,000571
1024	9,52	195,69	0,000537
2048	19,95	103,56	0,000507
3072	28,57	73,46	0,000476
4096	38,09	59,09	0,000444
5120	47,62	50,00	0,000420
6144	57,14	45,94	0,000381
7168	66,66	41,05	0,000365

Table 57: Results for AVBP, Test Case A on Bull x86 Cluster

7.15 ELMER

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	2996	0,000140
512	4,76	1246	0,000169
1024	9,52	672	0,000156
2048	19,95	384	0,000137
4096	38,09	264	0,000099

Table 58: Results for ELMER, Test Case A on Bull x86 Cluster

7.16 GPAW

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	2,38	75,78	0,0055427
512	4,76	55,26	0,0038005
1024	9,52	44,23	0,0023741
2048	19,95	33,36	0,0015738

Table 59: Results for GPAW, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
2048	6,96	1132	0,0001269
4096	13,93	702	0,0001023
8192	27,85	435	0,0000825
16384	55,71	307	0,0000585

Table 60: Results for GPAW, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
512	4,76	776,5	0,0002705
1024	9,52	515,3	0,0002038
2048	19,95	297,6	0,0001764

Table 61: Results for GPAW, Test Case B on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
8192	27,85	949,4	0,0000378
16384	55,71	538,4	0,0000333
32678	111,41	276,1	0,0000325
65536	222,82	156,5	0,0000287

Table 62: Results for GPAW, Test Case C on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1024	9,52	1842,2	0,0000570
2048	19,95	841,1	0,0000624
4096	38,09	417,4	0,0000629
8192	76,19	251,5	0,0000522

Table 63: Results for GPAW, Test Case C on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
20480	69,63	1845,1	0,0000078
40960	139,26	1014,8	0,0000071

Table 64: Results for GPAW, Test Case D on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
4096	38,09	2132,94	0,0000123
8192	76,19	1118,08	0,0000117

Table 65: Results for GPAW, Test Case D on Bull x86 Cluster

7.17 HELIUM

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
5253	17,86	2063	0,0000271
11781	40,06	1107	0,0000226
20910	71,09	2246	0,0000063

Table 66: Results for HELIUM, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
1540	14,32	195,5	0,000357
3003	27,93	110,4	0,000324

Table 67: Results for HELIUM, Test Case B on Bull x86 Cluster

7.18 OCTOPUS

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	0,87	187,28	0,006135
512	1,74	101,65	0,005651

Table 68: Results for OCTOPUS, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
128	1,19	106,51	0,007887
256	2,38	70,87	0,005927
512	4,76	51,21	0,004101

Table 69: Results for OCTOPUS, Test Case A on Bull x86 Cluster

7.19 PEPC

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
256	0,87	44,68	0,025714
512	1,74	22,92	0,025063
1024	3,48	11,95	0,024036
2048	6,96	6,55	0,021926
4096	13,93	3,90	0,018412
8192	27,85	2,67	0,013447
16384	55,71	2,31	0,007771
32678	111,41	2,75	0,003264
65536	222,82	4,41	0,001018

Table 70: Results for PEPC, Test Case A on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
2048	6,96	114,32	0,001256
4096	13,93	57,56	0,001247
8192	27,85	29,69	0,001209
16384	55,71	16,08	0,001116
32768	111,41	9,69	0,000926
65536	222,82	8,00	0,000561
131072	445,64	10,47	0,000214

Table 71: Results for PEPC, Test Case B on IBM BlueGene/P

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
16384	55,71	279,88	0,0000641
32678	111,41	140,96	0,0000637
65536	222,82	74,12	0,0000605
262144	891,29	36,36	0,0000309
294912	1002,7	37,50	0,0000266

Table 72: Results for PEPC, Test Case C on IBM BlueGene/P

7.20 SPECFEM3D

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
864	8,04	12960	0,0000096
1536	14,28	11841	0,0000051
2400	22,32	9787	0,0000046

Table 73: Results for SPECFEM3D, Test Case A on Bull x86 Cluster

No. of CPUs	Partition size (TFlop/s)	Time (sec)	Performance per Peak-TFlop/s
864	8,04	10585	0,0000118
1536	14,28	9182	0,0000076
2400	22,32	7665	0,0000056
3456	32,14	7808	0,0000040
4704	43,75	6329	0,0000036
6144	57,14	5509	0,0000032

Table 74: Results for SPECFEM3D, Test Case B on Bull x86 Cluster

7.21 TRIPOLI_4

No. of CPUs	Partition size (TFlop/s)	Equivalent Time (sec)	Performance per Peak-TFlop/s
32	0,30	7180	0,000468
64	0,60	3661	0,000459
128	1,19	2456	0,000342
256	2,38	1164	0,000361
512	4,76	652	0,000322

Table 75: Results for TRIPOLI_4, Test Case A on Bull x86 Cluster

8 Annex B

In this Annex the list of Benchmark Code Owners (BCOs) of PABS in PRACE 1IP can be found.

Benchmark Application	BCO
QCD	Stefanie Janetzko, FZJ
QUANTUM_ESPRESSO	Carlo Cavazzoni, CINECA
NAMD	Dimitris Dellis and Marios Chatziangelou, GRNET
CPMD	Carlo Cavazzoni, CINECA
CODE_SATURNE	Andrew Sunderland, STFC
GADGET	Xu Guo, EPCC
EUTERPE	Xavier Saez, BSC
WRF	Andrew Porter, STFC
NEMO	John Donners, SARA
CP2K	Dimitris Dellis and Marios Chatziangelou, GRNET
GROMACS	Dimitris Dellis and Marios Chatziangelou, GRNET
NS3D	John Donners, SARA
AVBP	Jacques David, CEA
HELIUM	Xu Guo, EPCC
TRIPOLI_4	Jacques David, CEA
PEPC	Stefanie Janetzko, FZJ
GPAW	Jussi Enkovaara, CSC
ALYA	Raul de la Cruz, BSC
OCTOPUS	Fernando Nogueira and Micael Oliveira, UC-LCA
BSIT	Raul de la Cruz, BSC
ELMER	Carlo Cavazzoni, CINECA
SPECFEM3D	Jacques David, CEA

Table 76: List of Benchmark Code Owner PABS